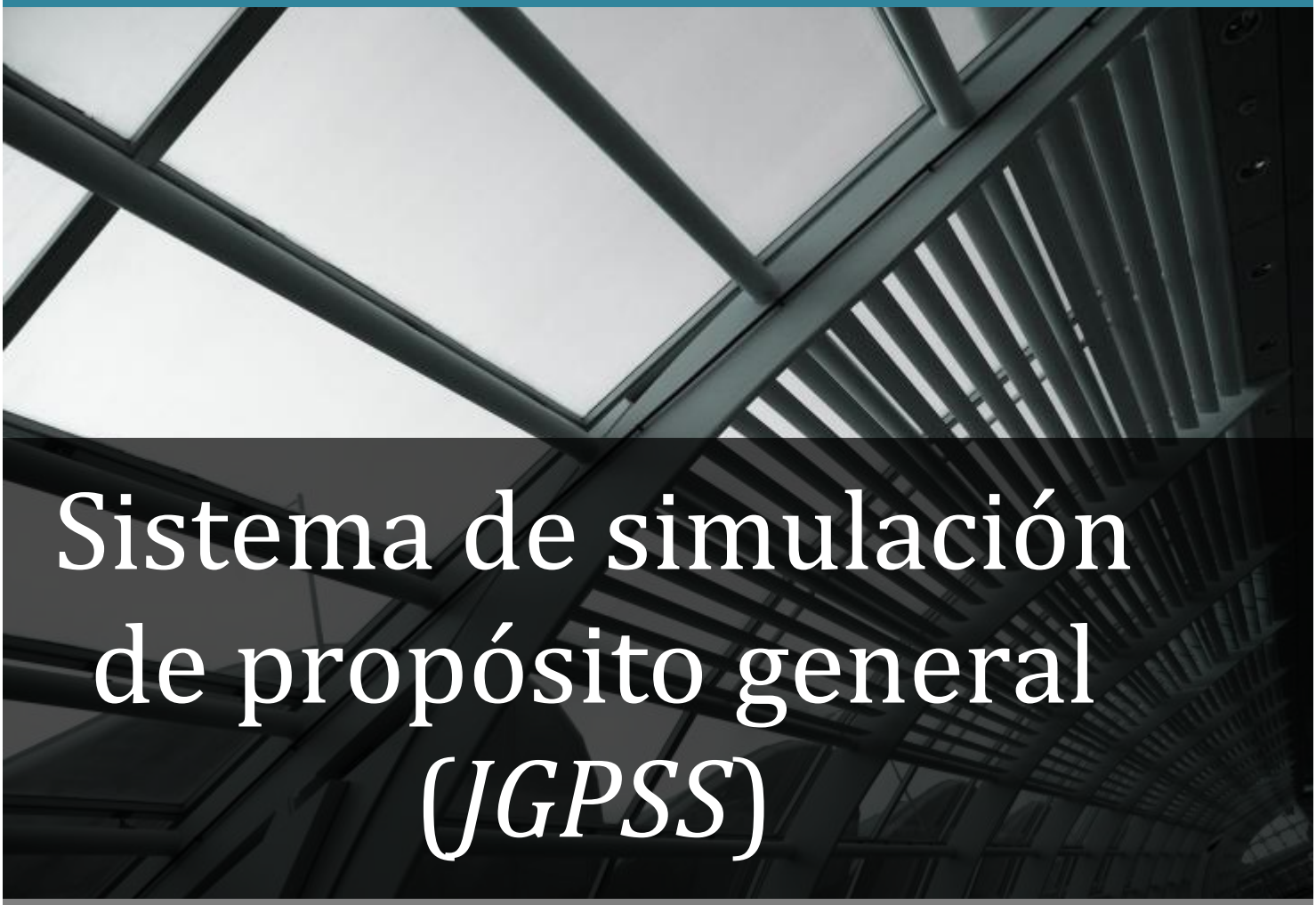


Sistema de simulación de propósito general (*JGPSS*)

Memoria

FACULTAD DE INFORMÁTICA DE BARCELONA

Alumno: Ezequiel Andujar Montes
Director: Pau Fonseca Casas
Especialidad: Ingeniería del Software
Fecha: 19/10/2017

Resumen

La simulación se ha convertido en una de las herramientas indispensables en nuestros tiempos y ha permitido modelar sistemas reales complejos para poder comprender el comportamiento de los mismos y poder mejorarlos.

Los grandes avances en tecnología han permitido mejorar cada vez más las técnicas de simulación, proporcionando herramientas cada vez más complejas y sofisticadas que requieren tiempo y dedicación para poder dominarlas.

Sin duda alguna, el número de profesionales requeridos para desarrollar modelos de simulación ha ido aumentando junto con la demanda de sistemas complejos que requieren un análisis detallado y exhaustivo para poder llevarse a cabo. Por este motivo es menester instruir de manera apropiada esta disciplina que es tan necesaria hoy en día.

Los tiempos actuales de que disponen los centros educativos y universidades, así como también las herramientas de simulación son muy limitados para poder abarcar todos los aspectos de la enseñanza en simulación.

Por este motivo el presente proyecto intenta abordar la eficaz enseñanza en simulación participando en el proyecto JGPSS, una herramienta *open source* basada en el lenguaje de simulación *GPSS* que tiene como objetivo el desarrollo de modelos de simulación de propósito general, así como también establecer las bases para convertir dicha herramienta en un producto comercial del que se puedan beneficiar tanto profesores y alumnos como profesionales del mundo de la simulación.

Abstract

Simulation has become one of the most essential tools of our times that has allowed us to create real complex system models in order to understand how these systems behave and how we can optimize them.

The continuous progress in technology has been the key to understand how the simulation techniques have been improved by developing such sophisticated and complex tools that requires a considerable amount of time to be mastered.

It is certainly known that the growth of complex systems have developed a high demand of computing professionals since they require a complex and exhaustive analysis in order to understand its inherent behavior so they can be improved. For this reason, it is essential for the simulation discipline to have well-formed professionals.

Educative centers and universities do not dispose nowadays the enough time in a semester period to successfully cover all the essential topics of simulation. For this reason, the present project attempts to cover the main aspects of simulation by building the JGPSS project, an open source tool based on the GPSS simulation language, that has as a main target the definition of general purpose system models as well as stablishing the bases to make this project a real commercial product which can be useful for the academic ambit as well as for the simulation professionals.

Glosario

1. **Framework**
   Infraestructura, armazón o marco. Define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

2. **Open Source**
   Código abierto. Es el software distribuido y desarrollado libremente. Se focaliza más en los beneficios prácticos (acceso al código fuente) que en cuestiones éticas o de libertad que tanto se destacan en el software libre.

3. **Stakeholder**
   En la gestión de proyectos, los involucrados o interesados. Son todas aquellas personas u organizaciones que afectan o son afectadas por el proyecto, ya sea de forma positiva o negativa. Una buena planificación de proyectos debe involucrar la identificación y clasificación de los interesados, así como el estudio y la determinación de sus necesidades y expectativas.

4. **Evento**
   Suceso que hace cambiar las variables de estado del sistema. Durante el procesamiento de un evento el tiempo de simulación permanece fijo. Un evento pertenece a una entidad, o actor en el sistema, y normalmente solo cambiara atributos de esta, dejando invariante el resto del sistema.

5. **Tiempo**
   Es el valor del tiempo que el simulador puede avanzar a una velocidad superior a la habitual de un reloj común, evolucionando así el estado de un sistema de forma acelerada.

6. **Modelo**
   Se puede entender como modelo de un sistema, toda la información que se tiene de las características y los componentes del mismo, de su estructura y comportamiento con respecto al medio que lo rodea y que permita crear una representación abstracta de este.

7. **Bug**
   Proviene del inglés. Literalmente "bicho", es un error o fallo en un programa informático o de un sistema software que desencadena un resultado indeseado.

8. **RNG**
   Random Number Generator. De la palabra inglesa número generador de números aleatorio.

Índice de contenido

ÍNDICE DE FIGURAS Y TABLAS

1. Introducción y estado del arte

1.1 Contextualización

El desarrollo del presente proyecto se realiza en calidad de trabajo de final de grado de los estudios en Ingeniería Informática en la especialidad de ingeniería del software y pretende abordar la etapa final de desarrollo del *framework JGPSS* [1] (*Java General Purpose Simulation System*).

JPGSS es un sistema *Open Source* de simulación de propósito general desarrollado íntegramente en el lenguaje de programación Java y que, a su vez, está basado en el lenguaje de simulación de propósito general a tiempo discreto ***GPSS*** (*General Purpose Simulation System*) desarrollado por el ingeniero Geoffrey Gordon a fines de la década de 1950.

El lenguaje de simulación *GPSS* no solo se ha utilizado durante años con el objetivo de realizar modelos de simulación de procesos generales[2], sino también como herramienta pedagógica en el mundo académico como soporte para la enseñanza de la simulación. Existen diversas herramientas basadas en la sintaxis de *GPSS*.

JGPSS está enfocado en gran medida como soporte a la enseñanza de la simulación. Su interfaz gráfica permite a los estudiantes construir modelos de simulación de una manera más intuitiva y visual. Además de brindar la posibilidad de obtener información estadística en diversos formatos sobre dichos modelos. El objetivo de dicho *framework* es permitir a los estudiantes de informática tener un conocimiento más profundo en la construcción de un completo motor de simulación de propósito general y así tener un conocimiento práctico sobre cómo funcionan los procesos internos involucrados en una herramienta de simulación. Otro de los objetivos de *JPGSS* es convertirlo en una herramienta al alcance no solo de estudiantes y profesores sino de cualquier persona que desee utilizarla como herramienta de aprendizaje o incluso ampliar las y/o añadir más funcionalidades al sistema. Finalmente, JGPSS puede llegar a convertirse en un producto comercial de gran valor para aquellas empresas que deseen o bien optimizar su productividad u ofrecer servicios el servicio del estudio de la productividad de empresas externas.

Como se ha comentado previamente el desarrollo del proyecto sigue la línea marcada por el *framework JGPSS* cuyo núcleo fue implementado por el profesor de la facultad de informática de Barcelona Pau Fonseca.

De esta manera se pretende utilizar dicho *framework* con el objetivo de ampliar y enriquecer las funcionalidades de las actuales implementaciones basadas en la sintaxis de *GPSS* que hay en el mercado.

[1] **JGPSS**: *Java General Purpose Simulation System*. Sistema de simulación de propósito general en Java

[2] **Proceso general:** Entendemos como proceso general, el conjunto de facetas de un objeto o sistema que le permite llevar a cabo una tarea específica. Como ejemplos tenemos desde el montaje de un coche en una cadena de producción hasta el funcionamiento de una cafetería.

Con dicho propósito será posible que los modelos de simulación desarrollados en *JGPSS* sean más visuales y fáciles de interpretar ya que se dispondrá de una interfaz gráfica más rica que ayudará y agilizará el proceso de construcción de modelos de simulación a la vez que facilitará la lectura y comprensión de los mismos.

Con ***JGPSS*** se pretende también que los resultados estadísticos de las ejecuciones de dichos modelos sean compatibles con la herramienta de *software R*[3] *y Excel* de estadística por tal de poder realizar un estudio más detallado de los resultados de la simulación, además de poder visualizarse los informes en texto plano o en formato PDF.

1.2 Actores implicados

Los *stakeholders* en este proyecto son los que se listan a continuación:

Desarrolladores

Los desarrolladores del proyecto son una de las principales partes interesadas: implicados en todo el proceso y directamente afectados por la evolución del desarrollo y por el resultado obtenido. En este caso concreto, el quepo de desarrolladores consiste en un único estudiante, el autor de este documento, ya que se trata de un trabajo de final de grado, que se realiza de forma individual y con carácter evaluativo.

Director

El profesor que dirige el TFG también tiene interés en la correcta realización del proyecto y se tendrá que implicar en todo el proceso. Su misión es guiar y supervisar el trabajo hecho por los desarrolladores. El director de este proyecto es Pau Fonseca i Casas, profesor del departamento de estadística e investigación operacional de la UPC.

Usuarios

Los usuarios de la herramienta desarrollada también serán beneficiarios del resultado del proyecto. Este conjunto se puede componer de desarrolladores de software, estudiantes y profesores de ingeniería del *software*, así como otros usuarios que tengan conocimientos del lenguaje GPSS o quieran aportar o ampliar funcionalidades del *framework.*

[3] **R** es un entorno y lenguaje de programación con un enfoque al análisis estadístico.

1.3 Estado del arte

JGPSS es una implementación del lenguaje de simulación *GPSS* el cual pretende ser una herramienta de simulación por eventos discretos y cuya finalidad es extraer información valiosa de un modelo diseñado con su sintaxis por tal de optimizar un proceso determinado.

La simulación por eventos discretos es una técnica informática de modelado dinámico de sistemas. Esta se caracteriza por un control en la variable del tiempo que permite avanzar a éste a intervalos variables, en función de la planificación de ocurrencia de tales eventos a un tiempo futuro. Un requisito para aplicar esta técnica es que las variables que definen el sistema no cambien su comportamiento durante el intervalo simulado.

Los primeros intentos para simular sistemas de eventos discretos, datan de la década de los años 60, donde se desarrollan las primeras simulaciones en ordenador para planear proyectos de gran envergadura, aunque a un costo alto y utilizando lenguajes de propósito general (a menudo *FORTRAN*). Las primeras herramientas para facilitar el uso de la simulación de evento discreto aparecen en la forma de lenguajes de simulación en la década de los años 70, aunque la programación en estos lenguajes se realiza todavía por medio de comandos escritos en un archivo. Lenguajes como *GPSS*, *SIMSCRIPT*, *SLAM* y *SIMAN* tienen una amplia difusión en los años 80, paralela a una gran producción científica relacionada con las posibles aplicaciones de la simulación de evento discreto, y el desarrollo de métodos para el análisis de experimentos por simulación, para generar por ordenador la ocurrencia de eventos siguiendo patrones probabilísticos, y para permitir que el motor del lenguaje pueda modelar una gama amplia de aplicaciones.

Actualmente se pueden distinguir en el mercado dos tipos de paquetes para simulación de evento discreto: los de propósito general y los orientados hacia alguna aplicación o sector industrial específico. Entre los paquetes más conocidos de propósito general, se pueden mencionar a *Arena*, *Simul8*, *GPSS/H*, *AweSim*, y *MODSIM III*, mientras que entre los paquetes con orientación hacia alguna aplicación se puede mencionar a *AutoMod*, *ProModel*, *SIMFACTORY II.5*, *QUEST* y *Arena Packaging Edition* para manufactura, *COMNET III* y *OPNET Modeler* para redes de comunicaciones, *SIMPROCESS*, *ProcessModel*, *ServiceModel* y *Arena Business Edition* para analizar flujos en procesos de negocios, y *MedModel* para servicios del cuidado de la salud. Los paquetes mencionados permiten la programación en un ambiente gráfico por medio de módulos, y pueden incorporar animación a sus modelos, lo que además de facilitar la programación del modelo de simulación, se constituye en una herramienta valiosa para la verificación y demostración de las capacidades del modelo.

Para hacer uso correcto de esta técnica y las herramientas citadas anteriormente es necesario contar con un profundo conocimiento sobre dichas herramientas. Sin embargo, esto presenta enormes desafíos en el ámbito académico, puesto que las herramientas profesionales son cada vez más complejas, los avances en investigación y desarrollo requieren que cada vez se incluyan más temas en los cursos de simulación, pero la duración de los cursos debe mantenerse acotada a uno o dos semestres. Por este motivo, en los últimos 20 años se han generado muchos proyectos de investigación con el objetivo de optimizar la enseñanza de simulación, mediante el diseño de entornos interactivos de enseñanza, la implementación de herramientas de simulación específicas para entornos de aprendizaje, y el diseño de marcos de trabajo (*frameworks*) para cursos de simulación. Dichos *frameworks* tiene que cumplir con las siguientes características: Facilidad de ingreso de datos, facilidad para interpretar la salida, facilidad para realizar replicaciones y experimentos, programación segura, eficiencia, disponibilidad y potencial avance hacia otras herramientas.

Son varias las propuestas de cumplen con las características mencionadas anteriormente. Ejemplo de estas son: *WebGPSS*, *WinGPSS*, *LinGPSS*, *MacGPSSS*, *FONWebGPSS*, ISDS, *GPSS*/*FON*, entre otras. La mayoría de ellas fueron diseñadas por los diferentes departamentos de universidades encargados de la enseñanza de simulación.

Si bien todas estas herramientas cumplen con las características que definen un marco de trabajo eficaz para la enseñanza, el proyecto *JGPSS* va un paso más allá y pretende ser una herramienta no solo de aprendizaje en el que el propio alumno pueda entender el funcionamiento del motor de simulación en detalle, sino que también sea capaz de modificarlo y crearlo el mismo y también poder dotarlo de funcionalidades extra por tal de satisfacer otros requisitos.

Podemos concluir, en consecuencia, que *JGPSS* es el único *framework* capaz de simplificar el desarrollo de una completa herramienta de simulación basada en la sintaxis *GPSS* y sobretodo una buena base de marco de trabajo para crear una herramienta de simulación más compleja que pueda satisfacer requisitos más amplios.

2. Alcance del proyecto

2.1 Objetivos

Los objetivos de este proyecto son por un lado completar el desarrollo de una herramienta de simulación a evento discreto de propósito general basada en la sintaxis del lenguaje de programación *GPSS* utilizando el *framework* JGPSS desarrollado para dicho propósito.

Por otro lado, el desarrollo de dicho proyecto servirá para ampliar y/o mejorar las funcionalidades que ofrecen actualmente en el mercado las otras implementaciones basadas en *GPSS*.

Uno de los principales aspectos en que se hará hincapié es en la mejora de los resultados estadísticos de la ejecución de los modelos de simulación, de manera que se consiga un formato de resultado compatible con herramientas software estadísticas como *R y Excel* por tal de conseguir un mayor entendimiento del funcionamiento del modelo. También será posible visualizar dichos informes en texto plano y en formato PDF.

Si bien existen muchas herramientas de simulación en el mercado capaz de realizar dichas funcionalidades, éstas son muy costosas y no es posible utilizarlas para uso didáctico debido a los altos costes de licencia, por tanto, es otro objetivo de este proyecto crear una herramienta *Open Source* capaz de realizar funcionalidades más completas y complejas y que esté al alcance estudiantes o usuarios que no quieran acarrear con costes altos de licencias.

Como objetivo secundario es interesante comentar que, al ser una herramienta *Open Source*, está podrá ser modificada y ampliada por los usuarios, de manera que todos puedan formar parte de la mejora de dicha herramienta. Para ello se utilizará una plataforma web con la que poder descargarse la herramienta.

Otro objetivo secundario será que sirva como herramienta didáctica para profundizar en el conocimiento del funcionamiento de un motor de simulación de propósito general en tiempo discreto.

Como último objetivo secundario, este proyecto tendrá la posibilidad de ser un producto comercial destinado a optimizar la productividad del sector empresarial.

2.2 Alcance

El alcance del proyecto estará limitado a un ámbito de proyecto de final de grado con posibilidades futuras de establecerse como un producto comercial que pueda competir con otras herramientas de alto coste. En este caso, los objetivos actuales del proyecto cubren las necesidades básicas de desarrollo de este trabajo de final de grado.
En consecuencia, se creará una herramienta de simulación de propósito general basada en la sintaxis GPSS con la que el usuario podrá crear o modificar modelos de simulación, recoger los resultados de la simulación en diversos formatos de manera que también puedan ser exportados a otras herramientas estadísticas de software y, además, poder visualizar la ejecución del modelo de manera gráfica para observar el comportamiento del modelo paso a paso.

La herramienta también será capaz de detectar errores de ejecución o de una mala definición del modelo, así como definir los modelos tanto gráficamente como con la sintaxis de GPSS de manera que se puedan evitar posibles errores por parte del usuario y así conseguir que el sistema sea más seguro.

2.3 Posibles obstáculos

La buena gestión del tiempo es uno de los principales factores de obstáculo del proyecto, puesto que está previsto realizarlo en el transcurso de cuatro meses, que es lo que dura el periodo del cuatrimestre universitario.

En consecuencia, será necesario realizar una planificación temporal realista y a la vez flexible, definiendo claramente qué casos de uso se implementarán en cada iteración, de manera que se pueda obtener un producto completo y funcional en el caso de que no se llegue a acabar en el plazo previsto.

Teniendo en cuenta que se utilizará un *framework* previamente implementado se tendrá que tener en cuenta los posibles *bug*s que éste pueda tener de manera que no suponga un gran impacto en el desarrollo del proyecto.

3. Metodología y rigor

3.1 Métodos de trabajo

En el desarrollo de este proyecto se hará uso de la metodología clásica con las fases en cascada estándares (análisis de requisitos, especificación, diseño e implementación del producto). Con el objetivo de garantizar un producto final funcional en caso de que no sea posible seguir la planificación temporal, se incluirán al menos tres iteraciones en el proceso.

Las fases de análisis, especificación y diseño no entran dentro del ámbito de este proyecto, por lo que el mismo se centra en la fase de implementación.

La primera iteración consistirá en implementar todos los bloques de GPSS que falten por implementar, de manera que se pueda crear modelos de simulación con todos los bloques que ofrece la sintaxis GPSS y realizar ejecuciones de dicho modelo sin errores.

Esta iteración es de vital importancia por tal de completar la herramienta de simulación satisfactoriamente. Para ello será necesario consultar exhaustivamente la documentación oficial [4] referente al funcionamiento de los bloques GPSS por tal de implementar correctamente su funcionamiento.

En la segunda iteración se realizará la mejora en los resultados de ejecución de los modelos de simulación, de manera que puedan ser visualizados no solo por la herramienta, sino también en diferentes formatos y que permita la exportación a otros programas estadísticos.

En la tercera iteración se manejarán los posibles errores que son lanzados por la herramienta y el control de dichos errores con el objetivo de que la herramienta no colapse.

Como parte importante de la metodología de trabajo a aplicar, es necesario destacar la comunicación frecuente con el director del proyecto.

Se harán, a ser posible, reuniones semanales o quincenales con el director del proyecto por tal de comprobar que se sigue con la planificación temporal y que el desarrollo de la herramienta se ajusta a los requisitos.

3.1.1 Herramientas de seguimiento

Se trabajará con una herramienta de control de versiones, junto con un gestor de repositorios en red (*Git* y *Github*), por tal de garantizar la disponibilidad del código y facilitar la recuperación de fallos. Para la comunicación con el director del proyecto se utilizarán herramientas convencionales habituales como *e-mail* y un sistema de compartición de ficheros en la nube como *Google Drive*.

[4] Implementación basada en la guía *Minutman corp*. http://www.minutemansoftware.com/reference/r7.htm

3.1.2 Método de validación

Con motivo de garantizar la validez y el cumplimiento de los requisitos tanto funcionales como de calidad se realizarán reuniones regulares con el director del proyecto.

Se garantizará una comunicación constante que facilitará la resolución rápida de dudas y problemas relacionados con el *framework JGPSS* que puedan ir surgiendo a lo largo del desarrollo de la herramienta.

3.2 Detalle de implementación de los bloques

Para poder realizar la implementación de los bloques, JGPSS define una clase abstracta llamada *Bloc*[5] cuyo método abstracto *execute* es el que implementará cada bloque. Dicho método será el responsable de acoger a las transacciones que lo atraviesan para generar información estadística y redirigir la transacción al siguiente bloque.
También se define un método abstracto *test*, que permitirá saber si una transacción puede o no atravesar el bloque en un momento determinado. Solo se especificará la implementación de este método, para aquellos bloques que tengan una lógica diferente para tratar este aspecto, ya que la mayoría de bloques admiten transacciones sin restricciones.

```
public abstract class Bloc {

    public abstrac void execute (Xact tr) throws Exception;
    public abstract boolean test (Xact tr);

    public Bloc nectBloc(Xact tr) {
              …
    }
}
```

Los bloques extenderán de la clase Block e implementarán el método *execute*. A continuación, se detallan en código, los bloques que JGPSS define para implementar.

3.2.1 Generate

```
@Override
public Bloc execute(Xact tr) throws Exception {

    if ((creationLimit && creationLimitNumber > 0) || !creationLimit) {

        Float evalA = Float.valueOf(evaluate(A, getModel(), tr));
        Float evalB = Float.valueOf(evaluate(B, getModel(), tr));

        Xact xact;
        xact = new Xact();
        xact.setBloc(this);
        xact.setProces(getProces());
        xact.setCreatTime(getModel().getRelativeClock());
        xact.setPriority(E);
        xact.setAssemblySet(VarGlobals.currentAssemblySet);

        if (getModel().getRelativeClock() == 0 && F > 0) {
            xact.setMoveTime(C);
        } else {
            xact.setMoveTime(getModel().getRelativeClock() + getGna().generate(evalA, evalB));
        }
        getModel().incIdXact();
        xact.setID(getModel().getIdxact());
        getModel().getFEC().add(xact);
        creationLimitNumber--;
        incTrans(xact);
        System.out.println("Generate DONE.");
    } else {
        System.out.println("Creation limit reached.");
    }
    return nextBloc(tr);
}
```

Figura 1. Implementación del bloque Generate

El bloque *Generate* es el encargado de generar las transacciones que circularán por el modelo. Los parámetros A y B son los usados para generar el tiempo de movimiento de

[5] Ver Anexo *Arquitectura del modelo de simulación*

las transacciones según una variable aleatoria (GNA) definida previamente en la creación del bloque. Si el parámetro C está definido, el tiempo de movimiento de las transacciones quedarán definidos por este. El parámetro E define la prioridad que tiene la transacción respecto a las Future Event Chain[6] (FEC) y Current Event Chain (CEC)[7].
La creación de transacciones viene limitada por el flag *crationLimit.*
Las transacciones generadas por este bloque se añadirán directamente a la FEC.

3.2.2 Advance

```
@Override
public Bloc execute(Xact tr) throws Exception {

    incTrans(tr);

    Float evalA = Float.valueOf(SNA.evaluate(A, getModel(), tr));
    Float evalB = Float.valueOf(SNA.evaluate(B, getModel(), tr));

    if (tr.getParameter("residual-time") != null) {
        tr.setMoveTime(getModel().getRelativeClock() + (Float) tr.getParameter("residual-time"));
    } else {
        tr.setMoveTime(getModel().getRelativeClock() + getGna().generate(evalA, evalB));
    }
    if (tr.getBlockRoute() != null) {
        tr.setBloc(tr.getBlockRoute());
    } else {
        nextBloc(tr);
    }

    getModel().getFEC().add(tr);
    return null;
}
```

Figura 2. Implementación del bloque Advance

El bloque *Advance* avanzará el tiempo de la transacción entrante según el GNA especificado. En caso de que la transacción tenga asignada una ruta específica, el próximo destino de la transacción será dicha ruta, en otro caso será el siguiente bloque de la lista y la transacción se añadirá a la FEC.

3.2.3 Terminate

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);
    getModel().setTC(getModel().getTC() - A);
    return null;
}
```

Figura 3. Implementación del bloque Terminate

El bloque *Terminate* decrementa el Transaction Counter[8] (TC) según el parámetro A y es donde se marca el final del modelo. Las transacciones se destruyen una vez que atraviesan este bloque.

[6] Future Event Chain. Consutar Anexo Esquema del Motor de simulación.
[7] Future Event Chain. Consutar Anexo Esquema del Motor de simulación.
[8] Transaction Counter. Consutar Anexo Esquema del Motor de simulación.

3.2.4 Seize

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);

    HashMap<String, Facility> facilities = getModel().getFacilities();

    if (facilities.get(A) == null) {
        facilities.put(A, new Facility(getModel()));
    }

    if (getModel().getBEC().get(A) == null) {
        getModel().getBEC().put(A, new PriorityQueue<>(1000, getModel().getPriorityComparator()));
    }

    // Attempt to capture the seize
    if (facilities.get(A).capture(tr)) {
        return nextBloc(tr);
    }

    // The Xacts remains on the Block event chain
    getModel().getBEC().get(A).add(tr);
    return null;
}
```

Figura 4. Implementación del bloque Seize

```
@Override
public boolean test(Xact tr) {

    HashMap<String, Facility> facilities = this.getModel().getFacilities();

    if (facilities.get(A) == null) {
        Facility fs = new Facility(getModel());
        facilities.put(A, fs);
    }

    boolean available = facilities.get(A).isAvailable();

    if (!available) {
        tr.setDelayed(true);
    }
    return available;
}
```

Figura 4.1 Implementación del método test del bloque Seize

El bloque *Seize* entra dentro de la categoría de *Facility*[9] que funciona como un recurso que solo puede utilizarse por una transacción a la vez, es decir, la primera transacción que entre en este bloque lo capturará de forma que ninguna otra transacción podrá entrar en él hasta que este sea liberado (ver bloque *Release*). Si una transacción intenta cruzar este bloque y este ya ha sido capturado por otra transacción, está se añadirá a la *Bloqued Event Chain*[10] (BEC), en caso contrario, seguirá avanzando al siguiente bloque.

Es posible que una transacción compruebe el estado del *Seize* antes de atravesarlo para averiguar si está disponible o no. En este caso el método *test* comprobará el estado de esta *Facility*.

[9] Facility. Estructura que permite gestionar el estado actual de los bloques que entran en esta categoría. I.E. Seize, Enter, Release, Leave, etc.

[10] Bloqued Event Chain. Estructura que almacena todas aquellas transacciones que están bloqueadas y no pueden seguir avanzando en el modelo.

3.2.5 Release

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);

    HashMap<String, Facility> facilities = getModel().getFacilities();

    facilities.get(A).release(tr);

    return nextBloc(tr);
}
```

Figura 5. Implementación del bloque Release

El bloque *Release* permite librera la *Facility* con el nombre dado por el parámetro A.

3.2.6 Queue

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);
    HashMap<String, QueueReport> queues = getModel().getQueues();

    if (queues.get(A) == null) {
        queues.put(A, new QueueReport(getModel()));
    }
    QueueReport queueStatistics = queues.get(A);
    queueStatistics.incCurrentCount(B);

    return nextBloc(tr);
}
```

Figura 6. Implementación del bloque Queue

El bloque *Queue* está relacionado con el bloque *Depart y QueueReport*[11] cuyo nombre viene definido por el parámetro A. La transacción que atraviesa este bloque aumentará el contador actual en B unidades del *QueueReport*. De esta manera es posible obtener datos como el tamaño de cola de transacciones que se ha formado, el tiempo medio cola, etc. (Ver QueueReport para ampliar información)

3.2.7 Depart

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);

    HashMap<String, QueueReport> queues = getModel().getQueues();

    queues.get(A).decCurrentCount(B);

    return nextBloc(tr);
}
```

Figura 7. Implementación del bloque Depart

El bloque Depart marca el fin de la zona del QueueReport con el nombre definido en el parámetro A. La transacción que atraviesa este bloque decrementará en B unidades el contador actual de QueueReport.

[11] QueueReport. Estructura que permite generar estadísticas de las transacciones que atraviesan la región comprendida entre los bloques *Queue* y *Depart*.

3.2.8 Enter

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);

    HashMap<String, Facility> facilities = getModel().getFacilities();

    Storage storage = getModel().getStorage(A);

    if (facilities.get(A) == null) {
        facilities.put(A, new Facility(getModel(), storage.getValor()));
    }

    if (getModel().getBEC().get(A) == null) {
        getModel().getBEC().put(A, new PriorityQueue<>(1000, getModel().getPriorityComparator()));
    }

    if (facilities.get(A).capture(B, tr)) {

        return nextBloc(tr);

    } else {
        getModel().getBEC().get(A).add(tr);
        return null;
    }
}
```

Figura 8. Implementación del bloque Enter

El bloque *Enter* funciona de manera análoga al *Seize*, es una *Facility* que permite ser capturada por las transacciones que la atraviesan. A diferencia del Seize, Enter dispone de un número de *Tokens*[12] que son capturados por las transacciones. De esta manera las transacciones que atraviesan el bloque Enter van capturando los *Tokens* disponibles hasta que ya no queden más. Esta característica supone que más de una transacción pueda capturar la *Facility* al mismo tiempo. Se podría decir que el bloque Seize es un tipo bloque *Enter* con un solo *Token* disponible.

Los *Tokens* están implementados como un contador de tipo entero, que se decrementando o incrementando a medida que las transacciones los van capturando o liberando. La capacidad máxima de Tokens que un bloque Enter puede tener, está definida en una entidad llamada *Storage*, que almacena la capacidad máxima de los bloques Enter.

Las transacciones que atraviesan este bloque capturan B Tokens en caso de que queden disponibles y pasan al siguiente bloque. Si no quedan Tokens por capturar, la transacción quede bloqueada y se añade a la BEC.

[12] Token: unidad de espacio de un Storage

3.2.9 Leave

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);

    getModel().getFacilities().get(A).release(B, tr);

    return nextBloc(tr);
}
```

Figura 9. Implmentación del bloque Leave

Las transacciones que atraviesan el bloque *Leave*, cuyo Facility tiene el nombre definido por el parámetro A, liberan B unidades de *Tokens* y pasan al siguiente bloque.

3.2.10 Assign

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);
    HashMap<String, Object> transactionParameters = tr.getTransactionParameters();

    if (transactionParameters.get(A.split("(\\+|\\-)")[0]) == null) {
        transactionParameters.put(A,B);
    } else {

        Float value = (Float) transactionParameters.get(A);

        if (A.endsWith("+")) {
            transactionParameters.put(A.split("\\+")[0], value + B);
        } else if (A.endsWith("-")) {
            transactionParameters.put(A.split("\\-")[0], value - B);
        } else {
            transactionParameters.put(A,B);
        }
    }
    return nextBloc(tr);
}
```

Figura 10. Implementación del bloque Assign

El bloque Assign está relacionado con los Transaction Parameters[13], cuya función es almacenar valores de tipo Real o String. Cuando una transacción atraviesa este bloque aplicará una operación al parámetro de la transacción cuyo identificador es el parámetro A del bloque. Dependiendo del operador que esté definido en el parámetro B (+ o -) se incrementará el valor del parámetro A en B unidades o se decrementará en B unidades. Une vez aplicada la operación, la transacción continuará al siguiente bloque.

[13] Transaction Parameters: Parámetros de transacciones. Las transacciones disponen de una serie de parámetros que pueden ser utilizados a lo largo de la ejecución del modelo.

3.2.12 Favail

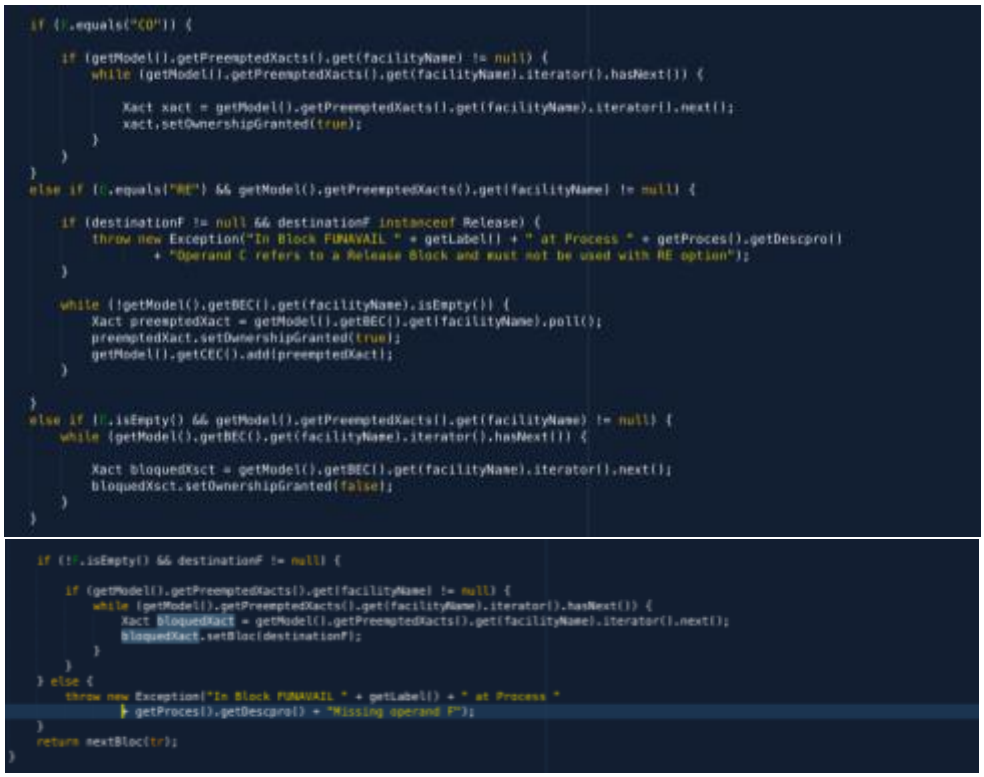
```
@Override
public Bloc execute(Xact tr) throws Exception {

    incTrans(tr);
    String facilityName = evaluate(A, getModel(), tr);

    PriorityQueue<Xact> BEC = getModel().getBEC().get(facilityName);
    PriorityQueue<Xact> preemptedXacts = getModel().getPreemptedXacts().get(facilityName);
    PriorityQueue<Xact> CEC = getModel().getCEC();

    if (getModel().getFacilities().get(facilityName) == null) {
        throw new FacilityNotFoundException();
    }

    if (getModel().getFacilities().get(facilityName).isAvailable()) {

        if (preemptedXacts != null && !preemptedXacts.isEmpty()) {
            CEC.add(preemptedXacts.poll());

        } else if (BEC != null && !BEC.isEmpty()) {
            CEC.add(BEC.poll());
        }
    }

    getModel().getFacilities().get(A).setAvailable(true);
    return nextBloc(tr);
}
```

Figura 11. Implementación del bloque Favail

El bloque *Favail* se asegura de que la entidad Facility, cuyo nombre se define en el parámetro A, este disponible para ser capturada. Si la Facility estaba previamente disponible, el bloque *Favail* intentará que la primera transacción que esté en la cola *PreemptedXacts* se añada a la CEC para que esta pueda capturar la Facility. En caso de que no haya transacciones en la cola *PreemptedXacts*, se procederá de la misma manera para la BEC.

3.2.13 Funavail

```
@Override
public Bloc execute(Xact tr) throws Exception {

    incTrans(tr);
    String facilityName = evaluate(C, getModel(), tr);
    Facility facilityState = getModel().getFacilities().get(facilityName);
    facilityState.setAvailable(false);

    Xact owningXact = facilityState.getOwningXact();

    Bloc destinationB = getProces().findBloc(evaluate(C, getModel(), tr));
    Bloc destinationF = getProces().findBloc(evaluate(F, getModel(), tr));

    if (O.equals("RE")) {

        if (destinationB != null && destinationB instanceof Release) {
            throw new Exception("In Block FUNAVAIL " + getLabel() + " at Process "
                    + getProces().getDescpro() + "Operand C refers to a Release Block and must not be used with RE option");
        }

        facilityState.setOwningXact(null);

    }
    else if (O.equals("CO")) {

    }
    else if (O.isEmpty()) {

        if (getModel().getPreemptedXacts().get(facilityName) == null) {

            getModel().getPreemptedXacts().put(facilityName, new PriorityQueue<>(1000, getModel().getPriorityComparator()));
        }
        getModel().getPreemptedXacts().get(facilityName).add(tr);
    }

    if (!F.isEmpty() && destinationB != null) {

        if (getModel().getFEC().contains(owningXact)) {
            for (Object b : getProces().getBlocs()) {

                if (b instanceof Advance) {
                    owningXact.setBloc((Bloc) b);
                    break;
                }
            }
            owningXact.setBlockRoute(destinationB);
        } else {
            owningXact.setBloc(destinationB);
        }
    } else {
        throw new Exception("In Block FUNAVAIL " + getLabel() + " at Process "
                + getProces().getDescpro() + "Missing operand C or block not found");
    }

    if (getModel().getFEC().contains(owningXact)) {

        String residualTimeName = evaluate(F, getModel(), tr);

        if (F.isEmpty()) {
            tr.restore(true);
        }

        float residualTimeValue = Math.abs(getModel().getRelativeClock() - owningXact.getMoveTime());

        if (!residualTimeName.isEmpty()) {
            owningXact.getTransactionParameters().put(residualTimeName, residualTimeValue);
        }

        owningXact.getTransactionParameters().put("residual-time", residualTimeValue);
        getModel().getFEC().remove(owningXact);
        owningXact.setMoveTime(owningXact.getMoveTime() - residualTimeValue);
        getModel().getBEC().get(0).add(owningXact);
    }
```



```
if (!.equals("CO")) {

    if (getModel().getPreemptedXacts().get(facilityName) != null) {
        while (getModel().getPreemptedXacts().get(facilityName).iterator().hasNext()) {

            Xact xact = getModel().getPreemptedXacts().get(facilityName).iterator().next();
            xact.setOwnershipGranted(true);
        }
    }
}
else if (!.equals("RE") && getModel().getPreemptedXacts().get(facilityName) != null) {

    if (destinationF != null && destinationF instanceof Release) {
        throw new Exception("In Block FUNAVAIL " + getLabel() + " at Process " + getProces().getDescpro()
                + "Operand C refers to a Release Block and must not be used with RE option");
    }

    while (!getModel().getBEC().get(facilityName).isEmpty()) {
        Xact preemptedXact = getModel().getBEC().get(facilityName).poll();
        preemptedXact.setOwnershipGranted(true);
        getModel().getCEC().add(preemptedXact);
    }

}
else if (!.isEmpty() && getModel().getPreemptedXacts().get(facilityName) != null) {
    while (getModel().getBEC().get(facilityName).iterator().hasNext()) {

        Xact bloquedXsct = getModel().getBEC().get(facilityName).iterator().next();
        bloquedXsct.setOwnershipGranted(false);
    }
}
```

```
if (!.isEmpty() && destinationF != null) {

    if (getModel().getPreemptedXacts().get(facilityName) != null) {
        while (getModel().getPreemptedXacts().get(facilityName).iterator().hasNext()) {
            Xact bloquedXact = getModel().getPreemptedXacts().get(facilityName).iterator().next();
            bloquedXact.setBloc(destinationF);
        }
    }
} else {
    throw new Exception("In Block FUNAVAIL " + getLabel() + " at Process "
            + getProces().getDescpro() + "Missing operand F");
}
return nextBloc(tr);
}
```

Figura 12. Implementación del bloque Funavail

La complejidad del bloque Funavail se debe a 3 clases de trasacciones con las que tiene que tartar.

- Transacciones que hayan capurado una *Facility* (Parámetros B y D)
- Transacciones en la cola *Preempted* (Parámetros E y F)
- Transacciones en la cola BEC (Parámetros G y H)

El bloque Funavail permite poner una Facility, identificada por A, en estado no disponible y controlar el destino de las transacciones que están esperando o usando la *Facility.*
Las transacciones que lleguen a este bloque durante el periodo de no disponibilidad serán bloqueadas y se les podrá otorgar la captura de la *Facility*. Si esta ya no estaba disponible, *Funavail* no tendrá efecto.

Cuando se usa la opción *REmove*, las transacciones se quitarán de la Facility si se utiliza esta opción con para las transacciones bloqueadas (BEC). Por ejemplo, si el parámetro G es RE, entonces el parámetro H se tiene que usar para redirigir las transacciones (bloque destino). Si su valor, en cambio, es *COntinue*, las transacciones que estén circulando por el modelo, habiendo capturado la Facility, podrán seguir ocupándola, incluso cuando esta esté no disponible, en este caso las estadísticas de la *Facility* se ajustarán para incluir este tiempo.

Cuando se usa un destino alternativo de bloque, se moverán las transacciones de su contexto y se redirigirán al nuevo bloque. Las transacciones bloqueadas, que se controlan con los parámetros G y H, no pueden cambiar de destino sin usar la opción *REmove*.

La transacción que ha capturado la *Facility*, que se controla por los parámetros B y D, y las *Preempted Transactions*, que se controlan con los parámetros E y F, pueden permanecer en contención de la Facility y ser redirigidas a nuevas destinaciones. Para ello, se especifica un destino alternativo sin usar el correspondiente parámetro RE.

Si la opción RE no se usa en el parámetro B, cualquier transacción que posea la *Facility*, se moverán a la cola PreemptedXacts. Este tipo de transacciones no pueden abandonar los bloques *Assemble*, *Gather* o *Match*, o entrar en un bloque *Advance* hasta que la cola *PreemptedXacts* se vaciado.

3.2.14 Gate

```
@Override
public Bloc execute(Xact tr) throws Exception {

    incTrans(tr);
    Bloc nextBlock = null;
    Bloc blockB = getProces().findBloc(B);

    Facility facility = getModel().getFacilities().get(A);

    boolean gateType = a.equals(U)
            || a.equals(NU)
            || a.equals(SF)
            || a.equals(NSF)
            || a.equals(SE) || a.equals(NSE);

    if (!gateType) {
        throw new Exception("At Gate Block " + getLabel() + ", Unknown gate type " + gateType);
    } else {
        boolean test = a.equals(U) && !facility.isAvailable()
                || a.equals(NU) && facility.isAvailable()
                || a.equals(SF) && facility.storageFull()
                || a.equals(NSF) && !facility.storageFull()
                || a.equals(SE) && facility.storageEmpty()
                || a.equals(NSE) && !facility.storageEmpty();

        if (test) {
            nextBlock = nextBloc(tr);
        } else {
            nextBlock = blockB;
        }
    }
    return nextBlock;
}
```

Figura 13. Implementación del bloque Gate

Las transacciones que atraviesan este bloque y no superan el test, se dirigirán al bloque especificado por el parámetro B. Si pasan el test, pasarán al siguiente bloque.

3.2.15 Logic

```
@Override
public Bloc execute(Xact tr) throws Exception {

    incTrans(tr);

    ArrayList<LogicSwitch> switches = getModel().getSwitches();

    LogicSwitch logicSwitch = switches.stream()//
            .filter(ls -> ls.getName().equals(A))//
            .findFirst()//
            .orElse(null);

    if (logicSwitch == null) {
        throw new LogicSwitchNotFound(A);
    }

    switch(x) {
        case S:
            logicSwitch.setState(true);
            break;
        case R:
            logicSwitch.setState(false);
            break;
        case I:
            logicSwitch.invert();
            break;
    }

    return nextBloc(tr);
}
```

Figura 14. Implementación del bloque Logic

Las transacciones que entren en este bloque cambiarán el estado del *LogicSwitch*[14] identificado por el parámetro A. Si el parámetro X es igual a S (SET) el estado del *LogicSwitch* cambiará a SET. Si el parámetro X es igual a R (RESET) el estado del *LogicSwitch* cambiará a RESET. En caso de que el parámetro sea igual a I, se invertirá el estado del LogicSwitch (si estaba a SET se pondrá a RESET y viceversa). La transacción que atraviesa este bloque siempre irá al siguiente en todo caso.

3.5.16 Loop

```
@Override
public Bloc execute(Xact tr) throws Exception {

    incTrans(tr);

    String param = evaluate(A, getModel(), tr);

    if (tr.getParameter(param) == null) {
        throw new ParameterNotANumberException(param);
    }
    if (!(tr.getParameter(param) instanceof Integer)) {
        throw new ParameterNotFoundException(param);
    }

    Integer counter = (Integer) tr.getParameter(param);

    counter -= 1;

    if (counter == 0) {
        tr.getTransactionParameters().put(param, counter);
        return nextBloc(tr);
    }

    return getProces().findBloc(B);
}

@Override
public String name() {
    return "Loop";
}
}
```

Figura 15. Implementación del Bloque Loop

[14] LogicSwitch: entidad que representa un interruptor que tiene dos estados: SET/RESET

El bloque *Loop* actúa como un bucle para las transacciones que lo atraviesan. Se evalua el parámetro A del bloque y se comprueba que exista como parámetro de la transacción. En caso de que este no exista o la transacción no sea de tipo Entero, se lanza una Excepción. En otro caso se comprueba si el valor del parámetro ha llegado a 0. En caso de hacerlo, la transacción sigue al bloque siguiente, en otro caso su destino será el bloque indicado en el parámetro B.

3.5.17 Priority

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);
    tr.setPriority(A);
    getModel().getCEC().add(tr);
    nextBloc(tr);
    return null;
}
```

Figura 16. Implementación del bloque Priority

A las transacciones que atraviesen este bloque se les asignará la prioridad definida en el parámetro A que cambiará su orden en la CEC. De esta manera se re planificará su avance en el modelo.

3.5.18 Savail

```
@Override
public Bloc execute(Xact tr) {

    getModel().getFacilities().get(A).setAvailable(true);
    return nextBloc(tr);
}
```

Figura 17. Implementación del bloque Savail

Las transacciones que atraviesan este bloque cambiarán el estado de la *Facility* con el nombre A estado disponible y continuarán al siguiente bloque.

3.5.19 Sunavail

```
@Override
public Bloc execute(Xact tr) throws Exception {

    String facilityName = evaluate(A, getModel(), tr);

    if (getModel().getFacilities().get(facilityName) == null) {
        getModel().getFacilities().put(facilityName, new Facility(getModel()));
    }

    Facility facilityState = getModel().getFacilities().get(facilityName);
    facilityState.setAvailable(false);

    return nextBloc(tr);

}
```

Figura 18. Implementación del bloque Sunavail

Las transacciones que atraviesan este bloque cambiarán el estado de la *Facility* con el nombre A a estado no disponible y continuarán al siguiente bloque.

3.5.20 SaveValue

```
@Override
public Bloc execute(Xact tr) throws Exception {

    incTrans(tr);

    SaveValue sv = getModel().getSaveValue(A);

    if (sv == null) {
        throw new SaveValueNotFoundException(A);
    }

    float value = getModel().getSaveValue(A).getValue();

    if (B.endsWith("-")) {
        float sValue = Float.parseFloat(B.split("\\-")[0]);
        sv.setValue(value - sValue);
    } else if (B.endsWith("+")) {
        float sValue = Float.parseFloat(B.split("\\+")[0]);
        sv.setValue(value + sValue);
    } else {
        float sValue = Float.parseFloat(B);
        sv.setValue(sValue);
    }
    return nextBloc(tr);
}
```

Figura 19. Implementación del bloque SaveValue

Este bloque está relacionado con las entidades del mismo nombre *SaveValue*, cuya finalidad es almacenar valores de tipo Real. Cuando una transacción atraviesa este bloque, accede al *SaveValue* con identificador A, y modifica su valor con el valor del parámetro B. Dependiendo del operador al final del parámetro B (+ o -) se le aplicara un incremento o un decremento. La transacción finalmente continuará al siguiente bloque.

3.5.21 Split

```
@Override
public Bloc execute(Xact tr) throws Exception {

    Integer serialNumber = 1;
    Bloc destinationBlock = getProces().findBloc(B);

    if (destinationBlock == null) {
        destinationBlock = nextBloc(tr);
    }

    if (!C.isEmpty()) {
        serialNumber = (Integer) tr.getParameter(C);
        if (serialNumber == null) {
            tr.getTransactionParameters().put(C, 1);
        }
    }

    while (A > 0) {

        try {
            Xact newXact = tr.clone();
            tr.setBloc(destinationBlock);
            serialNumber++;
            if (!C.isEmpty()) {
                tr.getTransactionParameters().put(C, serialNumber);
            }
            getModel().getCEC().add(newXact);

        } catch (CloneNotSupportedException e) {
            throw new Exception("At Split block " + getLabel() + " Error on copy transaction ");
        }
        A--;
    }
    return destinationBlock;
}
```

Figura 20. Implementación del bloque Split

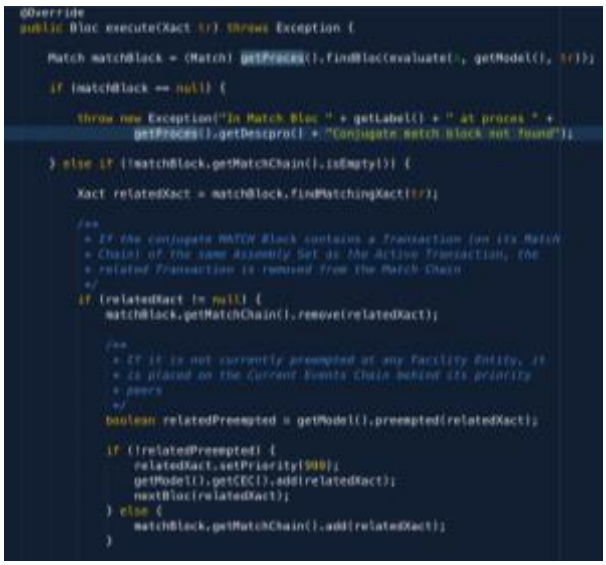
El bloque Split tiene como finalidad crear A clones de la transacción entrante a los cuales se les asignará un número de serie que se almacenará como parámetro de la transacción. Si el parámetro B está definido, la transacción será redirigida hacia el bloque B, en otro caso irán al siguiente bloque.

3.5.22 Test

Figura 21. Implementación del bloque Test

Las transacciones que entran en el bloque *Test* evalúan los parámetros A y B del bloque aplicándoles el operador X. Si la evaluación falla a la transacción no se le permite entrar en el bloque, por tanto, se añadirá a la FEC para volver a pasar por el bloque hasta que la evaluación sea cierta. En tal caso, si el parámetro C se ha especificado, la transacción continuará hasta el bloque especificado por el parámetro C, en otro coso continuará hasta el siguiente bloque.

3.5.23 Match



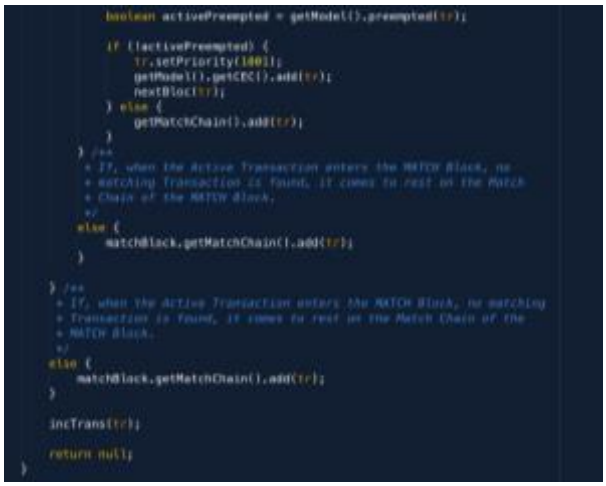


Figura 22. Implementación del bloque Match

Cuando una transacción entra en el bloque Match, se evalúa el parámetro A que identifica el bloque Match conjugado. Si no existe tal bloque se lanzará un error y la simulación se detendrá.

Si el bloque conjugado contiene una transacción en su *Match Chain*[15] del mismo conjunto de ensamblaje[16] que la transacción activa, la transacción asociada se quita de de la *Match Chain.* Si no se encuentra actualmente en la PreemptedXacts de alguna *Facility*, se desplazará a la CEC detrás de la prioridad de sus semejantes. De la misma manera, si la transacción activa no se encuentra en la *PreemptedXacts* de alguna *Facility*, se pondrá en la CEC, pero delante de sus semejantes.
Si alguna *Matching Transaction* está actualmente en la *PreemptedXacts* de una Facility, no se le permite salir del bloque Match hasta que todas las transacciones bloqueadas se hayan ido de su bloque Match conjugado.
Si en el momento en que la transacción activa entra en el bloque Match y no se encuentra ninguna transacción que haga *matching*, se añadirá a la *Match Chain* del bloque Match.

[15] Estructura en la que se almacenan las transacciones bloqueadas del bloque Match
[16] Es un número que identifica el conjunto de ensamblaje de cada transacción. Este es asignado en el momento de creación de la transacción en el bloque Generate.

3.5.24 Assemble

```
@Override
public Bloc execute(Xact tr) {

    incTrans(tr);

    Xact waitingXact = findWaitingXact(tr);

    if (waitingXact == null) {

        A--;

        tr.setCounter(A);
        if (A == 0) {
            return nextBloc(tr);
        } else {
            getMatchChain().add(tr);
        }
    } else {

        if (waitingXact.decCounter() == 0) {
            getMatchChain().remove(waitingXact);

            if (getModel().preempted(waitingXact)) {
                nextBloc(waitingXact);
                getModel().getCEC().add(waitingXact);
            }
        }
    }
    return null;
}
```

Figura 23. Implementación del bloque Assemble

Cuando una transacción entra en el bloque *Assemble*, se busca en la *Match Chain* del bloque una transacción a la espera del mismo conjunto de ensamblaje. Si no existen otros miembros del mismo conjunto de ensamblaje, se evaluará el parámetro A, se disminuirá en 1 unidad y se almacenará en la transacción. Si este número es igual a 0, la transacción intentará inmediatamente entrar en el siguiente bloque. En otro caso se añadirá a la *Match Chain* a la espera de otros miembros de su mismo conjunto de ensamblaje.

Cuando una transacción entra en el bloque Assemble, si se encuentra alguna transacción esperando, la transacción se destruye y el contador que se guardó en la transacción disminuirá en 1 unidad. Cuando este contador llegue a 0, la transacción que estaba esperando se quitará de la *Match Chain.* Si la transacción no se encontraba en la cola *PreemtedXacts* de ninguna *Facility,* intentará entrar en el siguiente bloque secuencial. Cuando lo haga, se planificará detrás de las transacciones activas con la misma prioridad.

Las transacciones que se encuentran en la *PreemptedXact* que completen un ensamblaje en cualquier bloque *Assemble*, no se les permite abandonar el bloque hasta que el resto se hayan marchado.

El bloque Assemble difiere del bloque Gather en que las transacciones que pasen con éxito son destruidas en el bloque Assemble.

3.5.25 Transfer

```
@Override
public Bloc execute(Xact tr) throws Exception {

    Bloc nextBlock = null;

    Xact activeTransaction = checkBloqued(tr);
    Bloc blocB = getProces().findBloc(B);
    Bloc blocC = getProces().findBloc(C);

    if (blocB == null) {
        blocB = nextBloc(tr);
    }
    if (blocC == null) {
        blocC = nextBloc(tr);
    }
```

Figura 24.1. Implementación del bloque Transfer (I)

El bloque Transfer puede operar en 9 modos, cada uno con diferentes propiedades. Cuando una transacción entra en este bloque, el parámetro A se usará para determinar el tipo de operación del bloque. El significado de los parámetros B y C dependerán del modo. Cuando no se especifica un parámetro que corresponda a la localización de un bloque, se usará el siguiente bloque secuencial después del bloque Transfer.

```
if (A.equals(BOTH)) {

    if (blocB.test(activeTransaction)) {
        nextBlock = blocB;
    } else if (blocC.test(activeTransaction)) {
        nextBlock = blocC;
    } else {
        activeTransaction.setDelayed(true);
        BloquedXacts.add(activeTransaction);
    }
} /**
```

Figura 24.2. Implementación del bloque Transfer (II)

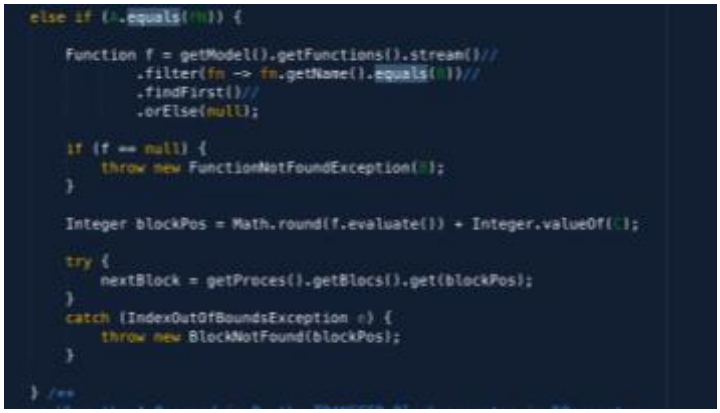
Cuando el parámetro A es igual a *BOTH*, el bloque Transfer operará en modo *Both*. En este modo, el bloque especificado por el parámetro B será comprobado (usando el método *test*). Si el bloque rechaza la transacción y no la admite, se comprobará el bloque especificado en el parámetro C. El primer bloque que admita la transacción, será el nuevo destino de esta. Si ninguno admite la transacción, se quedará en esperando en el bloque hasta que pueda entrar en algún bloque.

```
else if (A.equals(ALL)) {

    if (blocB.test(activeTransaction)) {
        nextBlock = blocB;
    } else {
        nextBlock = testBlocks(B, C, D, tr);

        if (nextBlock == null) {
            activeTransaction.setDelayed(true);
            BloquedXacts.add(activeTransaction);
        }
    }

} /**
```

Figura 24.3. Implementación del bloque Transfer (III)

Cuando el parámetro A es igual a *ALL*, bloque Transfer operará en modo *All.* En este modo el bloque especificado por el parámetro será testeado. Si este rechaza la transacción, los bloques serán testeados en serie hasta que bloque especificado por el parámetro C supere el test. A no ser que uno de los bloques intermedios admita la transacción antes de llegar el bloque C.
When the A Operand is ALL, the TRANSFER Block operates in "All Mode".

La localización de cada bloque que pase el test se calcula añadiendo el parámetro D a la localización del anterior bloque testeado. Si el operador D no se usa, todos los bloques entre los especificados por B y C, incluido, se testearán. Si el parámetro C no se usa, solo 1 bloque se testeará. El primer bloque que admita la transacción será el nuevo destino de esta. Si ningún bloque la admite, se quedará en el bloque Transfer hasta que pueda entrar en alguno.



```
else if (A.equals(FN)) {

    Function f = getModel().getFunctions().stream()//
            .filter(fn -> fn.getName().equals(B))//
            .findFirst()//
            .orElse(null);

    if (f == null) {
        throw new FunctionNotFoundException(B);
    }

    Integer blockPos = Math.round(f.evaluate()) + Integer.valueOf(C);

    try {
        nextBlock = getProces().getBlocs().get(blockPos);
    }
    catch (IndexOutOfBoundsException e) {
        throw new BlockNotFound(blockPos);
    }

} /**
```

Figura 24.4. Implementación del bloque Transfer (IV)

Cuando el parámetro A es FN, el bloque Transfer opera en modo *Function.* En este modo, el destino de la transacción es evaluada por la entidad *Function*, cuyo identificador se especifica en el parámetro B y se añade un incremento opcional especificado en el parámetro C.

```
else if (A.equals(SBR)) {

    tr.getTransactionParameters().put(C, new Float(getProces().getBlocs().indexOf(this)));

    if (blocB != null) {
        nextBlock = blocB;
    }

} /**
```

Figura 24.5. Implementación del bloque Transfer (V)

Cuando el parámetro A es SBR, el bloque Transfer opera en modo *Subroutine.* En este modo, la transacción activa siempre saltará a la localización especificada en el bloque B. La localización del bloque Transfer será almacenada en el parámetro de la transacción identificado por el parámetro C.

```
else if (A.equals(SIM)) {

    if (tr.isDelayed()) {
        nextBlock = blocC;
    } else {
        nextBlock = blocB;
    }
    tr.setDelayed(false);

} else if (A.equals(SNA)) {
```

Figura 24.6. Implementación del bloque Transfer (VI)

Cuando el parámetro A es SIM, el bloque Transfer opera en modo simultaneo. En este modo, la transacción activa saltará a una de las dos localizaciones, dependiendo del *flag Delay* de la transacción. Si este *flag* está activado, la transacción saltará a la localización especificada por el parámetro C y el *flag Delay* se reseteará. Si el *flag Delay* no está activado, la transacción saltará a la localización especificada por el parámetro B.

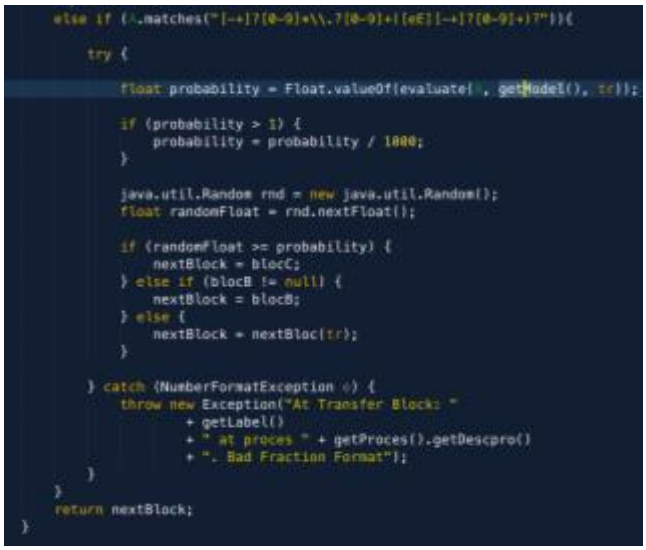


Figura 24.7. Implementación del bloque Transfer (VII)

Cuando el operando es un valor real, el bloque Transfer operará en modo *Fractional.* En este modo, la transacción activa salta a la localización especificada por el parámetro C con una probabilidad dada por el parámetro A (puede ser un SNA). Si el operando A es un número entero positivo, se interpretará como una fracción 1/1000 y se convertirá en una fracción de probabilidad. El destino alternativo se especificará en el parámetro B, en caso de que este no se especifique, el destino será el siguiente bloque secuencial.

3.3 Entidades implementadas

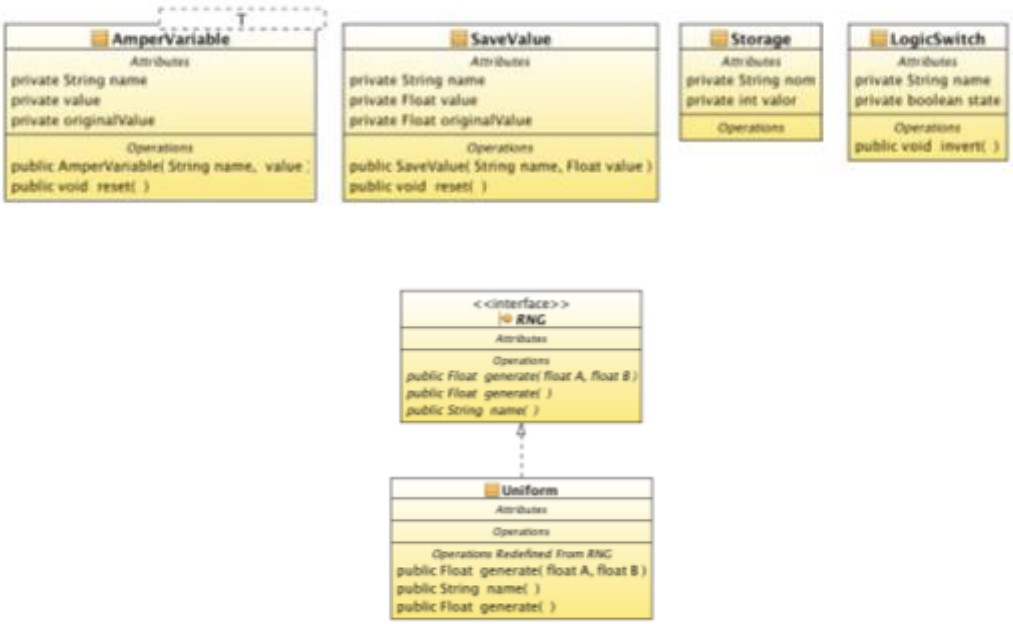


Figura 25. Diagrama de clases de las entidades

Las entidades que se muestran en la figura de arriba se han diseñado e implementado para las implementaciones de los bloques.

- AmperVariable: Almacena valores de tipo genérico que serán accedidos por las transacciones
- SaveValue: Almaceno valores de tipo Float que serán accedidos por las transacciones.
- Storage: Almacena la cantidad de *Tokens* que un bloque Server puede utilizar.
- LogicSwitch: Entidad que dispone de dos estados. Encendido o Apagado.
- RNG: define una interfaz que especifica dos métodos.
    - `Generate(Float A, Float B)` genera un valor aleatorio entre A±B
    - `Generate()` genera un valor aleatorio entre 0 y 1
- Uniform: Implementación de un RNG con una distribución uniforme.

3.4 Detalles de implementación de los Informes

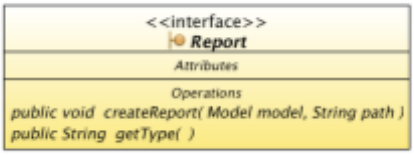


Figura 26. Diagrama de la interfaz Report

Para la generación de los informes se ha definido una interfaz que tendrán que implementar los diferentes tipos de Informes.

Las clases que implementan la interfaz *Report* son. PDFReport, CSVReport y TxtReport. A continuación, se mostrará parte de la implementación de las respectivas implementaciones.

3.4.1 PDFReport

```
@Override
public void createReport(Model model, String path) throws IOException, DocumentException {

    String fileName = path + "." + getType();
    Document document = new Document();
    PdfWriter.getInstance(document, new FileOutputStream(fileName));
    document.open();
    addMetaData(document, model);
    document.add(addTitlePage(model));
    document.add(addContent(model));
    document.close();
}
```

Figura 27. Implementación de PDFReport

3.4.2 CSVReport

```
@Override
public void createReport(Model model, String path) throws Exception {

    this.model = model;

    File file = new File(path + "." + getType());

    @Cleanup
    PrintWriter writer = new PrintWriter(file);

    printGeneralInfo(writer);
    printBlockInfo(writer);
    printFacilityInfo(writer);
    printQueueInfo(writer);
    printStorageInfo(writer);
    printSavesValues(writer);
    printCEC(writer);
    printFEC(writer);
}
```

Figura 28.Implementación de CSVReport

3.4.3 TxtReport

```
@Override
public void createReport(Model model, String path) throws Exception {

    this.model = model;

    File file = new File(path + "." + getType());

    @Cleanup
    PrintWriter writer = new PrintWriter(file);

    printGeneralInfo(writer);
    printBlockInfo(writer);
    printFacilityInfo(writer);
    printQueueInfo(writer);
    printStorageInfo(writer);
    printSavesValues(writer);
    printCEC(writer);
    printFEC(writer);
}
```

Figura 29. Implementación de TxtReport

Es similar a CSVReport, pero cambian las implementaciones de los métodos privados.

Por cuestiones de dimensiones, no se incluirán las implementaciones de los métodos privados de cada clase.

3.5 Mejoras realizadas

Las mejoras realizadas han consistido en refactorizar todo el código del *Framework*, actualizar las librerías obsoletas, como Java, a la última versión, añadir librerías nuevas que dan soporte al desarrollo (i.e Lombok[17], JUnit[18]) y optimizar métodos internos del framework para adaptarlos a las nuevas funcionalidades implementadas.

[17] https://projectlombok.org/
[18] http://junit.org/junit4/

Cambios añadidos

Se han realizado cambios en la interfaz gráfica, de manera que pueda soportar la creación de las diferentes entidades (*ver apartado 3.3*) y los diferentes tipos de informes. Concretamente se han añadido las vistas que se muestran a continuación.

Creación de la entidad *Function*

Figura 30. Vista para crear una Function

Creación de la entidad *SaveValue*

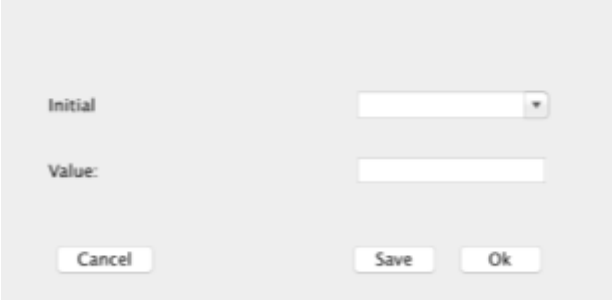


Figura 31. Vista para crear una SaveValue

Creación de la entidad *AmperVariable*

Figura 32. Vista para crear una AmperVariable

Creación de la entidad *Storage*

Figura 33. Vista para crear un Storage

Creación del tipo de informe

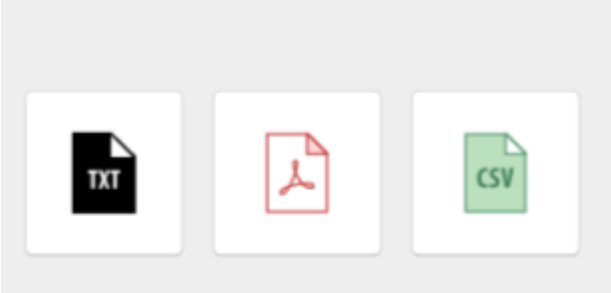


Figura 34. Vista para crear una Report

3.6 Test de funcionamiento y validación

Las pruebas funcionamiento y validación se han llevado a cabo, en primer lugar, utilizando la librería *JUnit* que permite realizar *tests* unitarios de las diferentes clases implementadas. Concretamente se realizaron *tests* unitarios de la entidad *Function* y la implementación *Uniform* de *RNG*. En útlimo lugar se especificaron una serie de modelos desarrollados con la herramienta para validar los resultados de los diferentes bloques y comprobar su correcto funcionamiento.

Implementación de los tests unitarios

```
1.  public class FunctionTest extends TestCase {
2.
3.  @Rule
4.  public ExpectedException thrown = ExpectedException.none();
5.
6.  public FunctionTest(String testName) {
7.  super(testName);
8.  }
9.
10.     @Override
11.     protected void setUp() throws Exception {
12.         super.setUp();
13.     }
14.
15.     @Override
16.     protected void tearDown() throws Exception {
17.         super.tearDown();
18.     }
19.
20.     @Test
21.     public void test1() throws Exception {
22.         Function f = new Function("test", "1", "D4", "1,3/2,5/3,8/4,12");
23.         assertEquals(4, f.getDistributionSize());
24.     }
25.
26.     @Test
27.     public void test2() throws Exception {
28.         Function f = new Function("test", "1", "C3", "1,3/2,5/3,8");
29.         assertEquals(3, f.getDistributionSize());
30.     }
31.
32.     @Test
33.     public void test3() throws MalformedFunctionDistributionException {
34.
35.         try {
36.             Function f = new Function("test", "1", "C6", "1,3/2,5/3,8");
37.         } catch (MalformedFunctionDistributionException e) {
38.             assertEquals(e.getMessage(), "Malformed function distribution");
39.         }
40.     }
41.
42.     @Test
43.     public void test4() throws MalformedFunctionDistributionException {
44.
45.         try {
46.             Function f = new Function("test", "1", "C3", "1,3/2,/3,8");
47.         } catch (MalformedFunctionDistributionException e) {
48.             assertEquals(e.getMessage(), "Malformed function distribution");
49.         }
50.     }
```

```
51.
52.     @Test
53.     public void test5() throws MalformedFunctionDistributionException {
54.
55.         try {
56.             Function f = new Function("test", "1", "C3", "1,3/2,r/3,8");
57.         } catch (MalformedFunctionDistributionException e) {
58.             assertEquals(e.getMessage(), "Malformed function distribution");
59.         }
60.     }
61.
62.     @Test
63.     public void test6() throws MalformedFunctionDistributionException {
64.
65.         try {
66.             Function f = new Function("test", "1", "X3", "1,3/2,5/3,8");
67.         } catch (MalformedFunctionDistributionException e) {
68.             assertEquals(e.getMessage(), "Malformed function distribution");
69.         }
70.     }
71.
72.     @Test
73.     public void test7() throws Exception {
74.         Function f = new Function("test", "1", "C3", "1,3/2,5/3,8");
75.         assertTrue(f.getB().startsWith("C"));
76.     }
77.
78.     @Test
79.     public void test8() throws MalformedFunctionDistributionException {
80.
81.         try {
82.             Function f = new Function("test", "1", "C03", "1,3/2,5/3,8");
83.         } catch (MalformedFunctionDistributionException e) {
84.             assertEquals(e.getMessage(), "Malformed function distribution");
85.         }
86.     }
87.
88.     @Test
89.     public void test9() throws Exception {
90.
91.         Function f = new Function("test", "0.4", "D5", ".4,1/.7,2/.85,3/.95,4/1,5"
    );
92.         assertEquals(f.evaluate(), 1f);
93.     }
94.
95.     @Test
96.     public void test10() throws Exception {
97.
98.         Function f = new Function("test", ".7", "D5", ".4,1/.7,2/.85,3/.95,4/1,5")
    ;
99.         assertEquals(f.evaluate(), 2f);
100.        }
101.
102.       @Test
103.       public void test11() throws Exception {
104.
105.           Function f = new Function("ts",".5","D5",".4,1/.7,2/.85,3/.95,4/1,5");

106.           assertEquals(f.evaluate(), 2f);
107.       }
108.
109.
110.
111.
112.
```

```
113.        @Test
114.        public void test12() throws Exception {
115.
116.            Function f = new Function("t",".96","D5",".4,1/.7,2/.8,3/.9,4/1,5");
117.            assertEquals(f.evaluate(), 5f);
118.        }
119.
120.        @Test
121.        public void test13() throws Exception {
122.
123.            Function f = new Function("test","1.","D5",".4,1/.7,2/.8,3/.9,4/1,5");
124.            assertEquals(f.evaluate(), 5f);
125.        }
126.
127.        @Test
128.        public void test14() throws Exception {
129.
130.            Function f = new Function("test", "1", "L4", "1,3/2,5/3,8/4,12");
131.            assertEquals(f.evaluate(), 3f);
132.        }
133.
134.        @Test
135.        public void test15() throws Exception {
136.
137.            Function f = new Function("test", "4", "L4", "1,3/2,5/3,8/4,12");
138.            assertEquals(f.evaluate(), 12f);
139.        }
140.
141.        @Test
142.        public void test16() throws Exception {
143.
144.            Function f = new Function("test", "20", "L4", "1,3/2,5/3,8/4,12");
145.            assertEquals(f.evaluate(), 0f);
146.        }
147.
148.        @Test
149.        public void test17() throws Exception {
150.
151.            Function f = new Function("te","RN1","D5",".4,1/.7,2/.8,3/.9,4/1,5");
152.            Float result = f.evaluate();
153.            assertTrue(result >= 1f && result <= 5f);
154.        }
155.
156.        @Test
157.        public void test18() throws Exception {
158.
159.            Function f = new Function("test", "RN5", "D5", "1,1/2,2/3,3/4,4/5,5");
160.            Float result = f.evaluate();
161.            assertTrue(result >= 1f && result <= 5f);
162.        }
163.    }
```

Figura 35. Implementación del test unitario de la entidad Function

Implementación de los tests de modelo

A continuación, se incluirán algunos *tests* de modelos realizados con la herramienta.

Test Bloque *Generate*

Modelo de simulación. Valor inicial del TC = 5

Figura 36. Definición de un modelo JGPSS (I)

Informe de la ejecución

JGPSS Model Report

generate test - to test the generate block

2017-10-23 12:16:25.219

General Model Information

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.0000	29.0543	2	0	0

Block Information

Proces: process

LABEL	LOC	BLOCK TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
gen	0	GENERATE	6	1	0
term	1	TERMINATE	5	0	0

CEC Information

XN	PRI	M1	ASSEM	CURRENT	PARAMETER	VALUES
6	0.0000	23.8242	1	0		

Figura 37. Informe de simulación en formato PDF (I)

Se puede apreciar que los resultados con coherentes con la definición del modelo y los parámetros de entrada.

Test bloques *Generate, Advance, Seize, Release* y *Terminate*

Modelo de simulación. Valor inicial del TC = 20

Figura 38. Definición de un modelo JGPSS (II)

Informe de la ejecución

JGPSS Model Report

Seize test - To test the Seize Block

2017-10-23 12:23:21.455

General Model Information

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.0000	87.2048	5	1	0

Block Information

Process: p1

LABEL	LOC	BLOCK TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	0	GENERATE	22	1	0
	1	SEIZE	21	0	17
	2	ADVANCE	21	0	0
	3	RELEASE	20	1	0
	4	TERMINATE	20	0	0

Facility information

FACILITY	ENTRIES	UTIL	AVE. TIME	AVAIL.	OWNER	INTER	DELAY
seize	21	0.9214751	3.826.3936	0	21	0	0

CEC Information

XN	PRI	M1	ASSEM	CURRENT	PARAMETER	VALUES
22	0.0000	84.0232	3	0		

FEC Information

XN	PRI	M1	ASSEM	CURRENT	PARAMETER	VALUES
21	0.0000	80.2281	3	5		

Figura 39. Informe de simulación en formato PDF (II)

Se puede apreciar que los resultados con coherentes con la definición del modelo y los parámetros de entrada.

4. Planificación Temporal

4.1 Planificación General

4.1.1 Planificación estimada del proyecto

Dada la naturaleza del tipo de proyecto que este se está abarcando y su condición de proyecto de final de grado, este tendrá una duración aproximada de 3 a 4 meses, tiempo de duración aproximado de un cuatrimestre de carrera. Serán contemplados al menos quince días de margen antes de la presentación para abordar posibles desviaciones que puedan surgir.

Fecha Inicial	Fecha Final	Duración aproximada
28/02/2017	30/06/2017	4 meses

Tabla 1. Descripción de las fechas inicial y final de proyecto y la duración aproximada total

4.1.2 Recursos empleados

A continuación, se describen los recursos empleados para la realización del proyecto.

Recurso	Tipo	Finalidad
Ordenador portátil MacBook Pro: Procesador 2,6GHz Intel Core i5 Memoria: 8GB 1600 MHz DDR3 Gráficos Intel Iris 1536Mb	Herramienta de desarrollo físico	Desarrollo de la aplicación Desarrollo de la documentación Desarrollo de la aplicación web
IDE *NetBeans*	Herramienta Software	Desarrollo de la aplicación.
Visual Studio Code	Herramienta Software	Desarrollo de la aplicación web
- Microsoft Word para Mac - Microsoft PowerPoint para Mac - Microsoft Office para Mac	Herramienta Software	- Desarrollo de la documentación - Desarrollo de la presentación
Git versión 2.5.4	Herramienta Software	- Control de versiones del repositorio del código fuente - Transferencia de ficheros hacia el servidor que alojará la aplicación web
Correo electrónico Hotmail	Herramienta Software	Comunicación con el ponente del proyecto
Servidor Web	Herramienta de desarrollo físico	Alojar la aplicación web de descarga del producto
Microsoft Project 2013	Herramienta Software	Realización la planificación del proyecto
Adobe Reader	Herramienta Software	Visualización documentos relacionados al proyecto
XAMPP versión 5.5.30	Herramienta Software	Realización pruebas locales sobre la aplicación web
Node.js	Herramienta Software	Framework que servirá la aplicación web del producto
Framework JGPSS	Herramienta Software	Framework base para desarrollar el producto

Tabla 2. Descripción de los recursos empleados en el proyecto

4.2 Tareas involucradas en el proyecto

Cabe destacar que es necesario tener conocimientos básicos sobre simulación y sobre el lenguaje que será la base del proyecto: *GPSS*.

Será de gran ayuda, por tanto, haber cursado la asignatura de Simulación impartida en la Facultad de Informática de Barcelona en la cual se introducen los aspectos teóricos básicos de la simulación, así como también la realización de diversas prácticas que servirán para familiarizarse con el lenguaje *GPSS* y la construcción del motor de simulación basado en dicho lenguaje e implementado con el *framework JGPSS*.

4.2.1 Primera fase de la gestión del proyecto

En esta fase se llevará a cabo la gestión del proyecto, que consistirá en realizar una serie de documentos que formarán parte de la memoria final del proyecto. En la primera fase de la gestión se definirá el problema que se intenta resolver mediante la realización de dicho proyecto, se definirá el alcance de este, así como también la planificación temporal que se llevará a cabo para organizar las diferentes fases de desarrollo de la aplicación.

Finalmente se realizará un estudio de la viabilidad económica con objeto de conocer los costes que conllevará y los beneficios que puede llegar a generar el producto una vez acabado. Dicha gestión se llevará a cabo gradualmente en una serie de entregas hasta llegar a completar el documento final con todas las especificaciones del proyecto.

4.2.2 Primer contacto con el Framework JGPSS

En esta primera fase será necesario familiarizarse con el *framework* de JGPSS mediante el cual se desarrollará todo el proyecto a lo largo del tiempo estimado. Como se ha mencionado previamente se trata de un *framework* escrito en el lenguaje de programación Java y forma parte de la base de la programación del proyecto, por lo tanto, es de vital importancia tener una base sólida de su estructura y funcionamiento, así como también es menester tener un conocimiento exhaustivo de la sintaxis del lenguaje de simulación *GPSS* con la finalidad de conocer los requisitos funcionales de éste y poder aplicarlo al proyecto. Se partirá de una versión más completa del *framework JGPSS* utilizado en la asignatura de Simulación.

Como dependencia de precedencia será necesario haber cursado la asignatura de Simulación como se ha comentado anteriormente o bien, tener conocimientos sobre teoría de simulación y el lenguaje *GPSS*. En este caso, se dará se ha realizado el curso de Simulación y se tienen dichos conocimientos. Se hará uso del manual de referencia del estándar *GPSS* para comprobar que las funcionalidades de los diferentes bloques que forman parte de la sintaxis de *GPSS* sean correctos.

4.2.3 Desarrollo

Puesto que se hará uso del *framework JGPSS*, las fases de especificación, análisis de requisitos y diseño ya han sido abordadas por el creador de dicho *framework*. Por tanto, se procederá a la implementación de las funcionalidades que ofrece el *framework* para, posteriormente, realizar mejoras y añadir más funcionalidades.

4.2.3.1 Implementación de los bloques GPSS

La sintaxis de GPSS consta de una serie de bloques o instrucciones que son utilizados para construir el modelo de simulación. En esta fase se procederá a implementar dichos bloques que están definidos en el *framework* JGPSS. Será de vital importancia realizar una serie de pruebas utilizando modelos de simulación básicos por tal de comprobar el correcto funcionamiento de cada uno de los bloques.

Una vez comprobado su correcto funcionamiento se procederá con el siguiente bloque hasta finalizar la implementación y la corrección de todos los bloques que forman parte de GPSS. Finalmente se diseñarán modelos más complejos sobre los cuales realizar pruebas exhaustivas para corroborar que funcione como está previsto.

4.2.3.2 Implementación de los informes de simulación

En esta fase se procederá a implementar los informes estadísticos que el programa de simulación debe presentar cada vez que se complete la ejecución con éxito de un modelo de simulación dado con unos parámetros concretos. Como se ha mencionado previamente, el programa de simulación debe ofrecer la posibilidad de presentar dichos informes en diversos formatos: informe en texto plano, informe compatible con aplicaciones basadas en hojas de cálculo (*Excel*), formato PDF y formato compatible con la herramienta software de estadística *R* .

4.2.3.3 Implementación de parámetros de la aplicación

En esta fase se implementará los posibles parámetros que se podrán introducir en la ejecución de un modelo de simulación de la aplicación, como por ejemplo los mensajes de error que pueda lanzar la aplicación, la generación del informe de simulación y el tipo de formato.

Cada uno de los parámetros se implementarán individualmente y se procederá posteriormente a realizar una serie de pruebas para comprobar su correcto funcionamiento.

4.2.4 Mejoras

Última fase sobre el desarrollo del producto que contempla mejoras en el funcionamiento, en el diseño o la adición de nuevas funcionalidades. El análisis y la planificación detallada de esta parte del proyecto se efectuará cuando se pueda comprobar de cuánto tiempo se dispone realmente y de cuáles son las aplicaciones necesarias. La dependencia de precedencia principal de esta etapa son las anteriores tareas.

4.2.5 Desarrollo de la aplicación WEB que alojará la descripción y características del producto

Esta fase consiste en desarrollar una aplicación web cuyo objetivo será describir JGPSS y las características que ofrece, así como también la posibilidad de descargarse la aplicación para escritorio y también mostrar un manual de uso de la aplicación.

4.2.6 Documentación y presentación

En esta fase se terminará la documentación y se revisará que sea correcta. Dicha documentación se irá haciendo a lo largo de las anteriores fases. A parte, se preparará la defensa del proyecto, que tendrá lugar en una fecha aún por concretar.

La memoria del proyecto incluirá la documentación redactada (y revisada) del módulo de GEP, la documentación derivada del análisis de requisitos, la especificación y diseño de cada fase y un manual de usuario del producto desarrollado. El final de la

documentación y la presentación final tienen como precedencia el resto de tareas del proyecto.

4.3 Plan de acción y valoración de alternativas

Existen una serie de factores que pueden afectar a la planificación del proyecto y su desarrollo normal.

Uno de los más importantes es el imprevisto temporal, que puede afectar a la estimación de las tareas. Es posible que una tarea tarde más de lo previsto en finalizarse de lo que se había estimado inicialmente y en consecuencia haga incrementar los costes de este y el presupuesto final cambie. En este caso será necesario incrementar el número de horas destinadas a esa tarea en concreto por tal de poder finalizarla como es debido. Esto tendrá un impacto en el coste del proyecto ya que son horas extras que se tendrán que renumerar.

A lo largo del desarrollo del proyecto se utilizarán herramientas *Software* de terceros que pueden fallar eventualmente y causar un impacto negativo en el desarrollo normal de las tareas. En caso de que una herramienta falle se tendrá que considerar el uso de una herramienta similar. Si, en última instancia no es posible dar solución al imprevisto, se tendrá que recurrir a una herramienta de pago que tendrá un impacto en el coste del proyecto.

Otro de los factores de riesgo es la avería del equipo que se está utilizando para desarrollar el proyecto. En caso de que no se disponga de garantía del mismo, será necesario comprar un equipo nuevo. Esto también afectará al coste del proyecto negativamente.

Es posible que surjan una serie de improvistos no planificados para los cuales será necesario aplicar un porcentaje de contingencia de manera que no tenga un impacto en el coste total de este. Para más información, consultar el apartado de estimación de costes.

5. Diagrama de Gantt

5.1 Tabla de tareas

Nombre de tarea		Horas
Gestión del proyecto fase 1	Definición, contexto y alcance del proyecto	18
	Planificación temporal	18
	Estudio de la viabilidad económica	21
	Presentación preliminar	19
Gestión del proyecto fase 2	Conjunto de condiciones	24
	Documento final	24
	Presentación final	24
Desarrollo	Implementación de los bloques GPSS	66
	Implementación de los informes de simulación	42
	Implementación de los parámetros de la aplicación	34
Mejoras		24
Desarrollo de la web que alojará el producto		32
Documentación y presentación		15
TOTAL		**361**

Tabla 3. Duración en horas de las tareas del proyecto

5.2 Diagrama

		Nombre	Duración	Inicio	Fin
1		**Gestión del proyecto fase 1**	20d?	**20/02/2017**	**17/03/2017**
2		Definición, contexto y alcance del proyecto	6d?	20/02/2017	27/02/2017
3		Planificación temporal	6d?	27/02/2017	06/03/2017
4		Estudio de la viabilidad económica	6d?	06/03/2017	13/03/2017
5		Presentación preliminar	5d?	13/03/2017	17/03/2017
6		**Gestión del proyecto fase 2**	9d?	**20/03/2017**	**30/03/2017**
7		Conjunto de condiciones	3d?	20/03/2017	22/03/2017
8		Documento final	3d?	23/03/2017	27/03/2017
9		Presentación final	3d?	28/03/2017	30/03/2017
10		**Desarrollo**	12d?	**30/03/2017**	**14/04/2017**
11		Implementación de los bloques GPSS	7d?	30/03/2017	07/04/2017
12		Implementación de los informes de simulación	4d?	07/04/2017	12/04/2017
13		Implementación de los parámetros de la aplicación	3d?	12/04/2017	14/04/2017
14		Mejoras	3d?	20/04/2017	24/04/2017
15		Desarrollo de la web que alojará el producto	4d?	25/04/2017	28/04/2017
16		Documentación y presentación	5d?	26/06/2017	30/06/2017

Ilustración 1. Tabla de tareas del diagrama de Gantt

Ilustración 2. Gráfica del diagrama de Gantt

6. Identificación de los costes

En esta sección se procederá a realizar un estudio sobre los costes del proyecto con la finalidad de realizar un presupuesto del mismo y poder estimar los beneficios que se esperan obtener una vez acabado el proyecto, siempre y cuando se decida su viabilidad.

Los costes del proyecto están relacionados con los recursos descritos en el apartado de planificación. En consecuencia, se tendrán en cuenta tanto los recursos humanos, así como también los recursos materiales *hardware* y *software*.

Se tendrán en cuenta los salarios medios reales de los diferentes miembros del equipo y se adaptarán a las horas asignadas a cada rol del proyecto, teniendo en cuenta la cantidad total de horas para realizar el proyecto.

A continuación, se describirán los costes del proyecto clasificándolos por recursos humanos y recursos materiales.

6.1 Recursos humanos

El proyecto será desarrollado por una sola persona que cumplirá los diferentes roles definidos para dicho proyecto. Dichos roles serán: el administrador de proyecto, analista desarrollador, desarrollador web y *tester*.

A continuación, se especifican los salarios reales por hora de los diferentes roles involucrados en el proyecto.

Rol	Precio/hora (€)
Administrador del proyecto	40
Analista desarrollador	35
Desarrollador Web	30
Tester	25

Ilustración 3. Costes de roles. Fuente extraída del proyecto: Software d'avaluació del rendiment de PCS (Oriol Gasset)

6.1.1 Costes directos por tarea

Una vez obtenidos los costes de los diferentes roles, se realiza el cálculo a nivel de tareas definidas en el apartado de planificación.
En la siguiente tabla se muestra el coste estimado para cada tarea según las horas dedicadas de cada rol.

Tarea	Tiempo de dedicación (en horas)				Coste estimado (€)
	Admin. proyecto	Analista desarrollador	*Tester*	*Desarrollador Web*	
Gestión del proyecto	148	0	0	0	5.920’00
Desarrollo	0	112	30	0	4.670’00
Mejoras	0	14	10	0	740
Desarrollo Web	0	0	0	32	960
Documentación y presentación	15	0	0	0	600
TOTAL	163	126	40	32	**12.890’00**

Tabla 4. Costes totales por tarea de los diferentes

En la siguiente tabla se puede apreciar los costes desglosados por rol.

Rol	Horas dedicadas	Precio por hora (€)	Coste estimado (€)
Administrador del proyecto	163	40	6.520’00
Analista desarrollador	126	35	4.410’00
Tester	40	25	1.000’00
Desarrollador Web	32	30	960’00
TOTAL	351		**12.890’00**

Tabla 5. Costes totales por roles

6.2 Recursos materiales

A continuación, se detallarán los costes de los recursos materiales que se clasificarán en dos tipos: *Hardware* y Software.

Para realizar el cálculo de dichos recursos se valorará la parte proporcional del precio de compra que se tiene que imputar en nuestro proyecto. Cabe destacar que el tiempo de vida, tanto de los recursos Hardware como Software es un factor a tener en cuenta, así como también la amortización de los mismos.

Para llevar a cabo el proyecto será necesario emplear una serie de componentes *Hardware* cuya vida útil según Hacienda es de 3 a 4 años y el tiempo de amortización es de 4 meses aproximadamente.

En cuanto a los componentes Software se asigna una vida útil de 2 a 3 años.
En la siguiente tabla se detallan los costes asociados a los recursos Hardware que se emplearán.

```
Fórmula de amortización = (Valor del bien / Tiempo de vida) * Tiempo de uso
```

Cabe destacar que para la mayoría de recursos Software que, utilizados, se hará uso de herramientas gratuitas que no supondrán ningún coste añadido al proyecto.

Producto	Tipo	Cantidad	Vida útil	Precio (€)	Amortización
MacBook PRO retina 13'	Hardware	1	4 años	1449'00	119'09
IDE Netbeans	Software	1	2 años	0	0
Visual Studio Code	Software	1	2 años	0	0
Microsoft Office for MAC	Software	1	2 años	149'00	24'69
Heroku service	Software	1	2 años	0	0
JGPSS Framework	Software	1	2 años	0	0
Node.js	Software	1	2 años	0	0
GIT v2.5.4	Software	1	2 años	0	0
TOTAL					**143'78**

Tabla 6. Costes de los recursos materiales

6.3 Otros gastos

Más allá de los costes mencionados, existen una serie de costes que no se tendrán en cuenta. Dichos costes se refieren a: consumo de agua, luz, coste de comida del personal, desplazamientos, etc. Puesto que el proyecto se desarrollará en un hogar particular no supondrán un impacto en el coste total del proyecto.

6.4 Costes Indirectos

6.4.1 Imprevisto temporal (I1)

Existe un riesgo de que no se pueda cumplir con la estimación temporal en el desarrollo del proyecto. En este caso será necesario incrementar el número de horas que dedicadas a realizar el desarrollo fuera de la estimación inicial. Dichas horas extra supondrán un incremento en el coste de los recursos del proyecto que tendrán un impacto en el presupuesto final. Se estima que las horas extra podrían suponer un 15% de las horas totales de desarrollo dedicadas al proyecto. Si realizamos el cálculo obtendremos que el coste podría aumentar en 598’00 €.

6.4.2 Imprevisto por avería (I2)

Existen imprevistos que también pueden incrementar el coste inicial estimado. Una posible avería con el equipo utilizado para desarrollar el proyecto podría aumentar el valor final del presupuesto si éste está fuera de garantía. Esto implicaría adquirir un equipo nuevo y calcular la parte proporcional amortizada que se sumaría a los costes inicialmente estimados. En este caso el valor calculado es, si el equipo se avería a mitad del proyecto, de 59’77 €.

6.4.3 Otros imprevistos (Contingencias)

Por último, es necesario aplicar un porcentaje de contingencia para todos aquellos imprevistos que no se hayan podido planificar. Esto permitirá un cumplimiento mucho más fiable del presupuesto final que en ningún caso se superará. Dicho margen de contingencia se establecerá en un 7%.

7. Estimación de los costes totales y viabilidad

Una vez identificados los costes tanto de los recursos humanos como de los recursos materiales, costes adicionales y costes indirectos, se calcula la estimación de costes totales del proyecto. A continuación, se puede apreciar el cómputo total de gastos desglosado por tipo de costes.

Tipo de coste		**Precio (€)**
Recursos humanos		12.890’00
Recursos materiales		143’78
Adicionales		0
Costes indirectos	Imprevisto temporal (I1)	598’00
	Imprevisto por avería (I2)	59’77
	Otros imprevistos (Contingencias)	942’33
TOTAL		**14.633’88**

Tabla 7. Estimación de los costes totales del proyecto

Una vez realizada la estimación de costes totales y teniendo en cuenta la planificación global y el cálculo de horas que se adapta al contexto de un proyecto de final de grado de 15 créditos ECTS, podemos asegurar la viabilidad del proyecto.

Cabe destacar que, si se tratase de un proyecto de empresa, se tendrían que tener en cuenta los diferentes costes derivados del consumo de luz, gas, agua, conexión a internet, herramientas software para empresas y licencias. A sí mismo, dichos costes se amortizarían con otros proyectos futuros, por tanto, se puede afirmar la viabilidad también para el caso de un proyecto real.

8. Control de gestión

Es importante definir un método eficaz para poder realizar un seguimiento detallado y constante del trabajo realizado por los recursos humanos empleados. De esta manera tendremos un margen de actuación para prevenir o en todo caso ajustar el presupuesto final en caso de que se produzcan imprevistos no deseados.

Se realizará un análisis de costes a final de cada tarea del proyecto, teniendo en cuenta las horas que se hayan dedicado a dicha tarea y haciendo una comparación con las horas que se hayan estimado para la misma.

En caso de que se produzca una diferencia notable, será necesario identificar las anomalías que haya podido causar esta diferencia y, en consecuencia, se ajustará las estimaciones del presupuesto para las siguientes fases de manera que se pueda realizar un presupuesto más ajustado a la realidad.

9. Informe de sostenibilidad

9.1 Económica

Factura estimada del proyecto

Tal y como se ha descrito en los apartados anteriores, se ha realizado una evaluación detallada sobre la estimación de los costes del proyecto, tanto a nivel de recursos humanos como de recursos materiales, así como también un ajuste de los costes sobre los posibles imprevistos que puedan surgir intentando limitar al máximo los valores de dichos costes.

Tipo	Recurso	Horas	Coste hora (€)	Coste Total (€)
Recursos humanos	Administrador del proyecto	163	40	6.520’00
	Analista desarrollador	126	35	4.410’00
	Tester	40	25	1.000’00
	Desarrollador web	32	30	960
Recursos materiales	MacBook PRO retina 13’			119’09
	IDE Netbeans			0
	Visual Studio Code			0
	Microsoft Office for MAC			24’69
	Heroku service			0
	JGPSS Framework			0
	Node.js			0
	GIT v2.5.4			0
Indirectos	Imprevisto temporal			598’00
	Implevisto avería			59’77
	Otros			942’33
TOTAL				**14.633’88**

Tabla 8. Factura del proyecto estimada

Factura final del proyecto

El proyecto finalmente ha supuesto una duración de 570 horas. Una diferencia de 229h con respecto a la estimación temporal inicial (361h).

Tipo	Recurso	Horas	Coste hora (€)	Coste Total (€)
Recursos humanos	Administrador del proyecto	180	40	7.200’00
	Analista desarrollador	250	35	8.750’00
	Tester	100	25	2.500’00
	Desarrollador web	40	30	1.200’00
Recursos materiales	MacBook PRO retina 13’			119’09
	IDE Netbeans			0
	Visual Studio Code			0
	Microsoft Office for MAC			55’10
	Heroku service			0
	JGPSS Framework			0
	Node.js			0
	GIT v2.5.4			0
TOTAL				**19.824’19**

Tabla 9. Factura final del proyecto.

Análisis de la estimación de costes vs. costes reales

Recursos	Incremento %	Incremento €
Administración del proyecto	9’44	680’00
Analista desarrollador	49’60	4.340’00
Tester	60’00	1.500’00
Desarrollador web	20’00	240’00
Recursos materiales	55’49	179’22
TOTAL		6.939’22

Tabla 10. Tabla comparativa costes estimados y reales.

En esta tabla vemos la diferencia entre los costes estimados y los costes reales del proyecto una vez finalizado. Podemos ver cuánto ha sido el aumento en valores porcentuales y monetarios de los diferentes recursos.

El total por imprevistos al finalizar el proyecto ha ascendido a 6.939’22 €. La estimación realizada para posibles costes por imprevistos se estimó en 1.600’10€. Por consiguiente, se llegó a cubrir un 23’06% de los imprevistos reales, quedando fuera de cobertura 5.190’31 € (26’18%).

Como podemos apreciar, la diferencia entre la estimación inicial y la final es notable, cosa que podemos atribuir a que la estimación de un proyecto es una técnica difícil de dominar y no siempre será precisa, puesto que hay imprevistos que escapan a nuestro control. En este caso se ha producido un aumento temporal significativo en la realización del proyecto. No obstante, ha supuesto una gran experiencia para aprender de este tipo de circunstancias que se dan en el ciclo de vida natural de todo proyecto software.

Cabe destacar que los costes del proyecto se han ajustado lo máximo posible, por tanto, es difícil una reducción de los mismos.

PLAN DE VIABILIDAD

Se estima que al menos existirá una actualización periódica del Software cada 3 meses aproximadamente y que ésta tendrá que realizarla un Analista desarrollador Java. La vida útil del Software diferirá dependiendo del uso que se le dé. No será lo mismo si se trata de la herramienta de una herramienta didáctica, cuya vida útil será más prolongada, que si se trata de un producto comercial que necesite de mantenimiento periódico.
Suponemos que la vida útil será en general un promedio de ambos escenarios: 5 años aproximadamente.

Estimamos que cada actualización/cambio del Software será de unas 6h promedio. Teniendo en cuenta que esto se realiza cada 3 meses y que el analista desarrollador tiene un coste de 35€/h, el coste estimado total durante la vida útil será de 4.200€.[19]

Valoración de la sostenibilidad económica

Teniendo en cuenta que la inversión inicial será de 19.824’19 € y dado los objetivos del proyecto y sus potenciales usuarios, se buscará cubrir dicha inversión en centros universitarios y/o de formación profesional, además del sector privado.

Es cierto que el proyecto conlleva una alta inversión inicial, no obstante, al tratarse de un Software de código abierto y sin costes de licencia, como ocurre en el caso de otros productos Software similares como aGPSS[20] (700$) o GPSS minute man [21](695$), puede llegar a ser a largo plazo más rentable, teniendo que cubrir solo los gastos por modificación/mejoras.

Por tanto, la puntuación es de **17**, ya que pese a no tener que cubrir los costes de licencia, la versión inicial sigue siendo elevada.

[19] `Coste total = vida útil x sesiones de actualización x tiempo actualización (horas)x tarifa desarrollador (€/h).`
[20] http://agpss.com/versions-sold.html
[21] http://www.minutemansoftware.com/product.htm

9.2 Social

Impacto personal

La realización de este proyecto supondrá en su hito inicial una primera puesta en contacto con la realización integral de un proyecto software y la posibilidad de aplicar y adaptar los conocimientos adquiridos a lo largo de la carrera, así como también de aprender y adquirir nuevos conocimientos a lo largo de la realización del mismo, lo que contribuirá a un crecimiento profesional que será útil a lo largo de la etapa laboral.

En su hito final he podido apreciar la complejidad de la realización de los proyectos software. La necesidad de realizar un buen trabajo previo, en cuanto la gestión inicial del mismo, una buena planificación y estimación son aspectos cruciales que tendrán un gran impacto en el proyecto final. Otro de los aspectos que debo destacar es la dificultad de realizar una estimación precisa temporal que tienen las diferentes fases del proyecto.

Puntuación: 6.

Impacto social

Este proyecto puede tener un impacto positivo dentro del colectivo de la enseñanza en simulación ya que, a diferencia de otras herramientas de simulación del mismo nivel, esta dispone de una interfaz gráfica mucho más visual e intuitiva para el alumno y, además, es más asequible que invertir en herramientas avanzadas y costosas. Otra de las facilidades que brinda la herramienta es la posibilidad de modificar y/o extender las funcionalidades, puesto que se trata de una herramienta *open source.*

Esto permitirá a los alumnos tener un mayor conocimiento de los motores de simulación de propósito general y acelerar la curva de aprendizaje en simulación, que es un aspecto que a medida que pasa el tiempo y aparecen nuevas herramientas más sofisticadas, se vuelve más complicado y costoso el trabajo de la enseñanza de esta disciplina.

Puntuación: 15.

Riesgos Sociales

Un posible riesgo podría ser la reducción de la plantilla en una empresa que haya optimizado su productividad utilizando esta herramienta.

Puntuación: -5.

9.3 Ambiental

Consumo del diseño

El principal recurso que se utiliza es un portátil Mac Book, cuyo consumo energético en kilo vatios por hora es de 17KW/h. Si tenemos en cuenta que la realización del proyecto ha supuesto unas 536h de desarrollo, el consumo energético total será de 17 x 536 = 9112 KW/h.

No obstante, como se trata de un recurso que se emplea para múltiples tareas independientemente de este proyecto, no supone un gran incremento de su consumo energético. Podemos suponer que la realización de este proyecto supone un tercio del consumo energético total del recurso empleado. Por consiguiente, el consumo energético total del proyecto será de 1/3 x 9112 KW/h = 3037,33 KW/h.

Para el desarrollo de este proyecto se han aprovechado otros recursos Software existentes que han supuesto un ahorro en tiempo y coste y que implica un menor impacto medioambiental.

El resto de recursos indirectos que se utilicen, como servicios de luz, gas o agua utilizados por las personas que desarrollan el proyecto (en este caso una), se emplearán de una manera responsable intentando no desperdiciar innecesariamente dichos recursos. Dichos recursos no serán contemplados como parte del consumo energético del proyecto.

Puesto que se trata de un proyecto Software, no existen residuos contaminantes de fabricación, ni emisiones de CO2 en gran exceso que puedan tener un impacto negativo en el medio ambiente.

Podemos concluir que el desarrollo del proyecto no implicará un impacto negativo para el medio ambiente.

Valoración: 10.

Huella ecológica

En cuanto a la huella ecológica que dejará el proyecto, principalmente será positiva ya que, si se consigue la optimización de la producción, se puede optimizar al mismo tiempo los recursos naturales.

Valoración: 20.

Riesgos ambientales

Los posibles escenarios que pudieran aumentar la huella ecológica derivado del uso del presente proyecto son prácticamente escasos. No obstante, se podría dar la situación de que los usuarios de la aplicación decidieran imprimir en papel los informes generados por la aplicación. Esto tendría un impacto negativo para el medio ambiente.

Valoración: -1.

9.4 Tabla de sostenibilidad

En la siguiente tabla se asigna la puntuación a los diferentes aspectos de la sostenibilidad del proyecto.

	Sostenibilidad			Total
	PPP	Vida útil	Riesgos	
Económico	5	12	0	17
Social	6	15	-5	16
Ambiental	10	20	-1	29
Total	21	47	-6	**62**

Tabla 9. Puntuación de sostenibilidad

9.5 Conclusiones

Como podemos extraer de las valoraciones sobre la sostenibilidad, con una puntuación de 62, podemos afirmar que el proyecto es sostenible. En primer lugar, porque su impacto medioambiental es irrelevante y además puede tener un impacto positivo en el medio ambiente ya que podría optimizar la utilización de recursos del sector privado. En segundo lugar, en el ámbito académico puede llegar a tener un gran valor, tanto para alumnos, como para profesores o cualquier persona que tenga interés en aprender simulación y también de mejorar el proyecto y añadir funcionalidades nuevas. Además, puede aportar un gran valor a aquellas empresas que busquen optimizar su productividad. Finalmente, en tercer lugar, el impacto más negativo es la sostenibilidad económica que reduce la puntuación global de sostenibilidad, ya que requiere de una inversión inicial elevada para llevarse a cabo.

10. Conclusiones finales

Para empezar, analizaremos el cumplimiento de los objetivos planteados inicialmente. Por una parte, los objetivos principales se han cumplido. En primer lugar, se ha conseguido implementar todos los bloques necesarios para realizar una simulación de modelos definidos con la interfaz gráfica. Esta tarea ha servido para profundizar en los conocimientos del lenguaje GPSS, aunque ha sido la que más tiempo ha llevado, a pesar de la estimación poco realista inicial. En segundo lugar, se ha conseguido que el programa sea capaz de generar los informes estadísticos de la ejecución del modelo. Esta tarea ha sido más sencilla de implementar que la anterior puesto que se recogían los datos ya calculados del modelo. Igualmente ha sido una tarea más tediosa, ya que para generar los informes en los diferentes formatos (TXT, PDF, Excel y R) se han tenido que tener en cuenta las particularidades de cada formato. En último lugar, se ha conseguido la creación de una plataforma web mediante la cual los usuarios podrán acceder a la herramienta y aprender a utilizarla mediante el manual de usuario. Esta tarea no ha supuesto gran esfuerzo ya que se han utilizado conocimientos previos para realizarla. Por otra parte, los objetivos secundarios de "Aportación de valor en la enseñanza en simulación" y "Producto comercial" no se podrán valorar en este punto del proyecto.

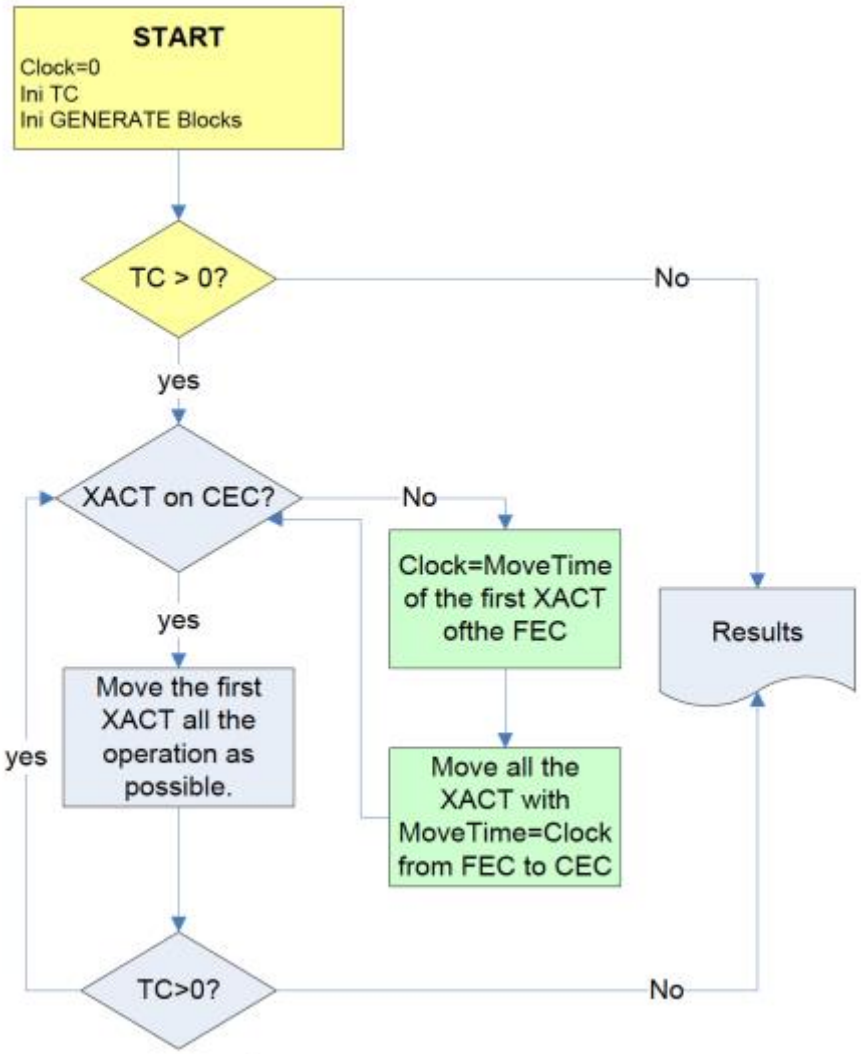
Con respecto a la planificación temporal, la estimación inicial no se ajustó a la realidad ya que se hizo una valoración excesivamente optimista de la duración que podía llegar a tener el proyecto. No obstante, se puede extraer de esta experiencia, que una planificación es una tarea difícil y requiere de un análisis exhaustivo y muchas variables a tener en cuenta que, dada la naturaleza del proyecto, no se ha podido cubrir.

En relación a la sostenibilidad del proyecto, se estimaba que fuera una herramienta más sostenible, pero los datos contradicen este hecho, debido a que, a nivel económico, la realización de la estimación de costes se hizo de manera optimista sin tener realmente en cuenta que el tiempo es un factor clave y difícil de estimar. Por tanto, esto ha supuesto un impacto negativo en la sostenibilidad global del proyecto, pese a ser social y medioambientalmente positivos.

Finalmente, la realización de este proyecto ha supuesto la realización integral de un proyecto software con la posibilidad de aplicar y adaptar los conocimientos adquiridos a lo largo de la carrera, así como también de aprender y adquirir nuevos conocimientos a lo largo de la realización del proyecto, lo que ha contribuido a un crecimiento profesional que será útil a lo largo de la etapa laboral.

11. Anexos

10.1 Esquema del Motor de simulación[22]



TC: *Transaction Counter*. Define la condición de finalización.
XACT: *Transaction*. Entidad de que circula por el modelo.
CEC: *Current Event Chain*. Estructura que almacena las transacciones actuales.
FEC: *Future Event Chain*. Estructura que almacena las transacciones futuras.

[22] Extracción del paper JGPSS, AN OPEN SOURCE GPSS FRAMEWORK TO TEACH SIMULATION (Pau Fonseca I Casas, Josep Casanovas)

10.2 Arquitectura JGPSS[23]

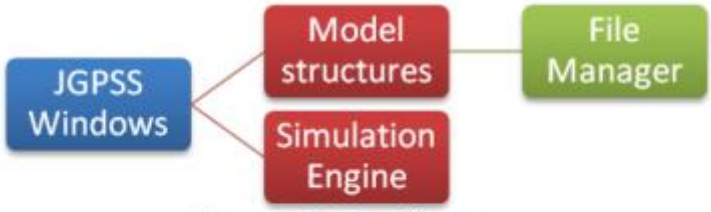
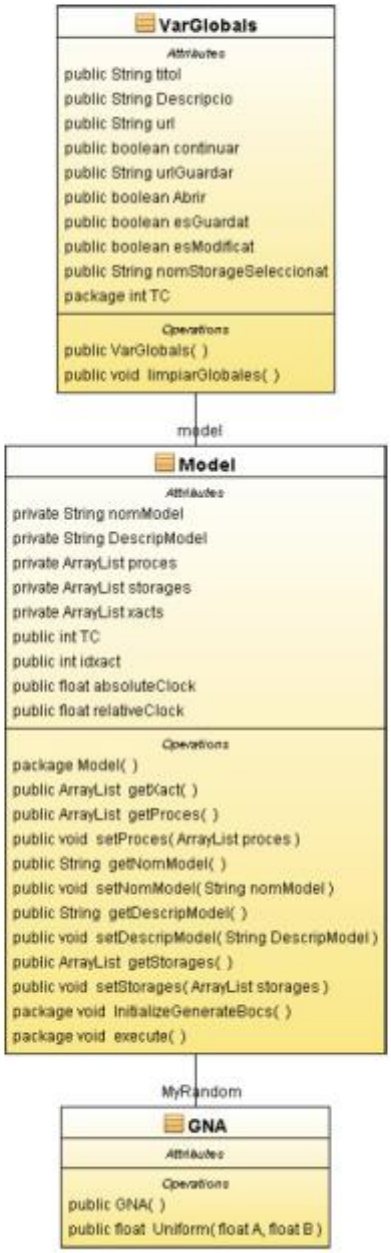
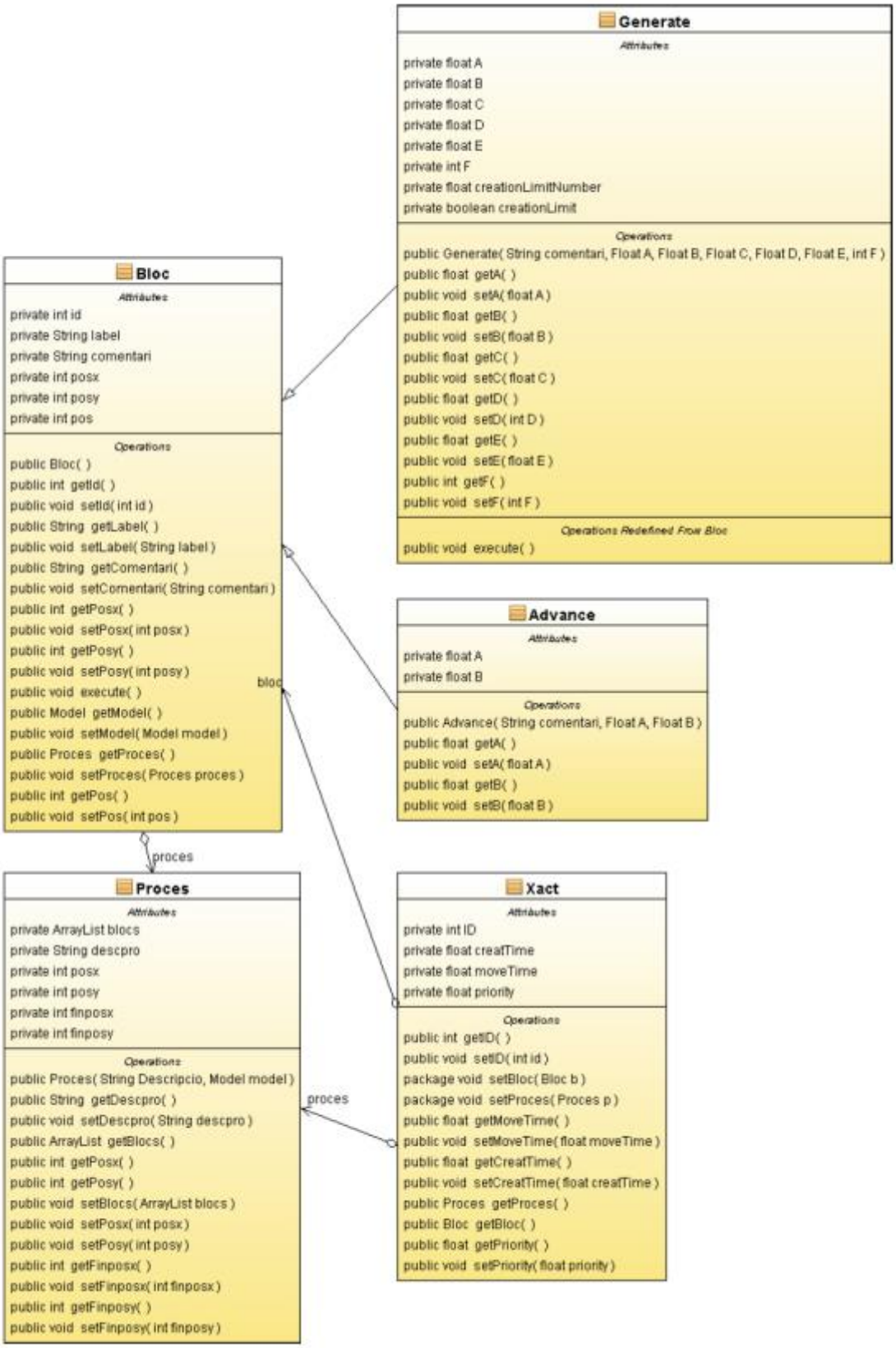


Figure 7: JGPSS architecture



[23] Extracción del paper JGPSS, AN OPEN SOURCE GPSS FRAMEWORK TO TEACH SIMULATION (Pau Fonseca I Casas, Josep Casanovas)


Generate
Attributes
private float A
private float B
private float C
private float D
private float E
private int F
private float creationLimitNumber
private boolean creationLimit
Operations
public Generate(String comentari, Float A, Float B, Float C, Float D, Float E, int F)
public float getA( )
public void setA(float A)
public float getB( )
public void setB(float B)
public float getC( )
public void setC(float C)
public float getD( )
public void setD(int D)
public float getE( )
public void setE(float E)
public int getF( )
public void setF(int F)
Operations Redefined From Bloc
public void execute( )
Bloc
Attributes
private int id
private String label
private String comentari
private int posx
private int posy
private int pos
Operations
public Bloc( )
public int getId( )
public void setId(int id)
public String getLabel( )
public void setLabel(String label)
public String getComentari( )
public void setComentari(String comentari)
public int getPosx( )
public void setPosx(int posx)
public int getPosy( )
public void setPosy(int posy)
public void execute( )
public Model getModel( )
public void setModel(Model model)
public Proces getProces( )
public void setProces(Proces proces)
public int getPos( )
public void setPos(int pos)
bloc
Advance
Attributes
private float A
private float B
Operations
public Advance(String comentari, Float A, Float B)
public float getA( )
public void setA(float A)
public float getB( )
public void setB(float B)
proces
Proces
Attributes
private ArrayList blocs
private String descpro
private int posx
private int posy
private int finposx
private int finposy
Operations
public Proces(String Descripcio, Model model)
public String getDescpro( )
public void setDescpro(String descpro)
public ArrayList getBlocs( )
public int getPosx( )
public int getPosy( )
public void setBlocs(ArrayList blocs)
public void setPosx(int posx)
public void setPosy(int posy)
public int getFinposx( )
public void setFinposx(int finposx)
public int getFinposy( )
public void setFinposy(int finposy)
proces
Xact
Attributes
private int ID
private float creatTime
private float moveTime
private float priority
Operations
public int getID( )
public void setID(int id)
package void setBloc(Bloc b)
package void setProces(Proces p)
public float getMoveTime( )
public void setMoveTime(float moveTime)
public float getCreatTime( )
public void setCreatTime(float creatTime)
public Proces getProces( )
public Bloc getBloc( )
public float getPriority( )
public void setPriority(float priority)

Bibliografía

[1] JGPSS, un marco de trabajo de código abierto para enseñar simulación /
Fonseca i Casas, P., Casanovas, J. (2009)
[*JGPSS, AN OPEN SOURCE GPSS FRAMEWORK TO TEACH SIMULATION*]

[2] Modelos de simulación de eventos discretos y de procesos continuos.
Fuente http://dinamica-de-sistemas.com/revista/0608o.htm

[3] Un entorno de aprendizaje y una propuesta de enseñanza de simulación de eventos discretos con GPSS / Luján Villareal, G. (2013)

[4] Gestión de proyectos.
Fuente: https://es.wikipedia.org/wiki/Gesti%C3%B3n_de_proyectos

[4] Simulación por eventos discretos.
Fuente https://es.wikipedia.org/wiki/Simulación_por_eventos_discretos

[6] Informe de Sostenibilidad del TFG
Fuente https://www.fib.upc.edu/sites/fib/files/documents/estudis/tfg-como-plantear-informe-esp.pdf

[7] GPSS World Reference Manual
Fuente http://www.minutemansoftware.com/reference/reference_manual.htm